Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

Department of mathematics
Study major: Mathematical Informatics

Mathematical modeling of phylogenetic compression

MASTER THESIS

Author: Veronika Hendrychová
Supervisor: Karel Břinda
Year: 2025/2026

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hendrychová** Jméno: **Veronika** Osobní číslo: **502475**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Matematická informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Matematické modelování fylogenetické komprese

Název diplomové práce anglicky:

Mathematical modeling of phylogenetic compression

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Karel Břinda, Ph.D. INRIA, the French National Research Institute for Computer Science and Automation

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

doc. Ing. Lubomíra Dvořáková, Ph.D. katedra matematiky FJFI

Datum zadání diplomové práce: **31.10.2025** Termín odevzdání diplomové práce: **08.05.2026**

Platnost zadání diplomové práce: **30.09.2027**

Zuzana Masáková
Digitálně podepsal(a) Zuzana Masáková
Datum: 25.11.2025 11:23:01
ID: 40553

podpis vedoucí(ho) ústavu/katedry

Milan Krbálek
Digitálně podepsal(a) Milan Krbálek
Datum: 25.11.2025 11:56:11
ID: 40556

podpis proděkana(ky) z pověření děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

26.11.2025

Datum převzetí zadání

Bc. Hendrychová Veronika

Podpis studentky

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hendrychová** Jméno: **Veronika** Osobní číslo: **502475**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Matematická informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Matematické modelování fylogenetické komprese

Název diplomové práce anglicky:

Mathematical modeling of phylogenetic compression

Pokyny pro vypracování:

1. Seznamte se s problémem prohledávání genomických databází ([1], [2]), s moderními milionovými databázemi bakteriálních genomů ([3]), a s technikou fylogenetické komprese [4].
2. Nastudujte dostupné protokoly fylogenetické komprese pro různé reprezentace genomů (assemblies, k-merové množiny) a různé druhy aplikací, včetně softwarových nástrojů (MiniPhy, Phylign atd.) [4].
3. Seznamte se s problematikou k-merových datových struktur ([5]), s moderními programy na nich postavenými (např. Fulgor) a s nízkoúrovňovými technikami komprese binárních matic.
4. Vytvořte matematický model fylogenetické komprese pro genomové reprezentace odpovídajících binárních matic (např. k-merová či bloom-filtrová matice) s použitím RLE jako nízkoúrovňové kompresní techniky.
5. Formulujte kompresi jako optimalizační problém a porovnejte jeho řešení pomocí fylogenetické komprese s optimálním řešením. Analyzujte časovou a prostorovou složitost jednotlivých řešení ([6]).
6. Zvolte vhodný model shlukové analýzy (clustering) a zahrňte ho do svého modelu fylogenetické komprese. Analyzujte vliv na teoretické a praktické výsledky modelu.
7. Experimentálně otestujte navržený model na reálných datech bakteriálních genomů a porovnejte dosažené výsledky komprese s optimálním řešením získaným pomocí TSP solveru [7].
8. Na základě experimentálních výsledků učiňte závěry o vhodnosti zvoleného modelu, efektivitě navrženého protokolu a diskutujte vhodnost jeho použití na různé typy dat (např. pro jeden nebo více bakteriálních druhů, velikost databáze atd.).

Seznam doporučené literatury:

[1] Y. W. Yu, N. M. Daniels, D. C. Danko, and B. Berger, “Entropy-scaling search of massive biological data,” Cell Syst, vol. 1, no. 2, pp. 130–140, Aug. 2015.
[2] P.-R. Loh, M. Baym, and B. Berger, “Compressive genomics,” Nat Biotechnol, vol. 30, no. 7, pp. 627–630, Jul. 2012.
[3] M. Hunt, L. Lima, W. Shen, J. Lees, and Z. Iqbal, “AllTheBacteria - all bacterial genomes assembled, available and searchable,” bioRxiv, Mar. 2024, doi: 10.1101/2024.03.08.584059.
[4] K. Břinda et al., “Efficient and robust search of microbial genomes via phylogenetic compression,” Nat Methods, vol. 22, no. 4, pp. 692–697, Apr. 2025.
[5] C. Marchet, C. Boucher, S. J. Puglisi, P. Medvedev, M. Salson, and R. Chikhi, “Data structures based on k-mers for querying large collections of sequencing data sets,” Genome Res., vol. 31, no. 1, pp. 1–12, Jan. 2021.
[6] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, The traveling salesman problem. in Princeton Series in Applied Mathematics, no. 17. Princeton, NJ: Princeton University Press, 2007.

FAKULTA JADERNÁ A FYZIKÁLNĚ INŽENÝRSKÁ
FACULTY OF NUCLEAR SCIENCES AND PHYSICAL ENGINEERING
Břehová 7
115 19 Praha 1

DECLARATION

I, the undersigned

Student's surname, given name(s):	Hendrychová Veronika
Personal number:	502475
Programme name:	Mathematical Informatics

declare that I have elaborated the master's thesis entitled

Mathematical modeling of phylogenetic compression

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 08.01.2026

Bc. Veronika Hendrychová
.....
student's signature

Acknowledgement

I am very grateful to my supervisor, Karel Břinda, for his guidance, support and enthusiasm about this topic, and for his thoughtful feedback that has shaped this thesis from the very beginning. I also thank Martin Vaněk and Ľubomíra Dvořáková for proofreading the work. I acknowledge the support of SGS 161-1612303D000.

Veronika Hendrychová

Title:
Mathematical modeling of phylogenetic compression

Author: Veronika Hendrychová

Study major: Mathematical Informatics
Type: Master thesis

Supervisor: Karel Břinda
Inria, French National Institute for Research in Digital Science and Technology
Consultant: Lubomíra Dvořáková
FJFI ČVUT

Abstract: Bacterial genome collections are growing faster than current computational capacities, presenting a fundamental scalability challenge. Recently, phylogenetic compression has demonstrated gains of one to several orders of magnitude in compressing and searching large and diverse bacterial genome collections by exploiting evolutionary relationships among genomes to guide algorithms and data structures. However, the mathematical principles underlying these gains remain insufficiently understood. Here, we introduce a formal framework that establishes the theoretical foundations of phylogenetic compression. It models genomic variation with binary matrices and run-length encoding (RLE), and formulates compression as an optimization problem. We show that, for agnostic data, this problem is NP-hard, via a reduction from the Traveling Salesperson Problem (TSP). Under simplified evolutionary models such as the Infinite Sites Model (ISM), phylogenetically guided compression admits polynomial-time optimal solutions, with tree-based heuristics, such as Neighbor Joining (NJ), provably achieving optimality. Our analyses of real bacterial genome datasets further show that NJ and similar algorithms achieve near-optimal compression in practice, even when ISM assumptions are only partially satisfied. Together, these results provide a rigorous theoretical foundation for phylogenetic compression and demonstrate its potential to substantially reduce storage and computational costs for large genomic datasets.

Keywords: phylogenetic compression, run-length encoding, computational biology, modeling

Contents

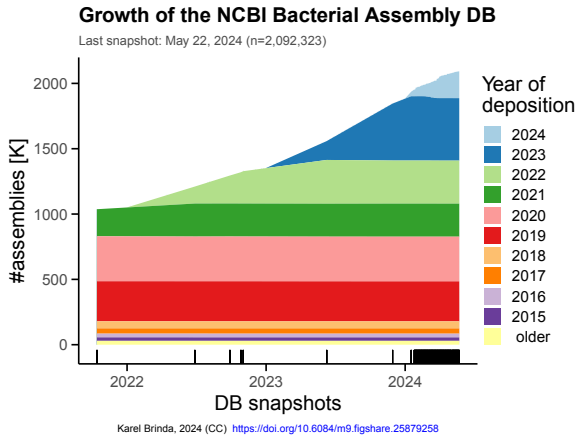
List of Figures

Chapter 1

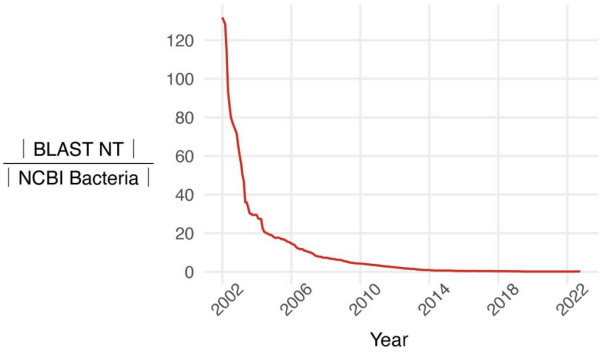
Introduction

DNA sequencing is a fundamental technology that converts the biological information encoded in DNA into digital data that form the basis for many areas of the life sciences. Over the past decades, the development of massively parallel sequencing technologies [1, 2] has made it possible to read DNA rapidly, accurately, and at a decreasing cost. As these technologies have advanced, databases of genomic sequences have become an essential resource in modern medicine, biology, biotechnology, epidemiology, and numerous related fields.

However, public repositories of genomic collections have been expanding at an exponential rate [3, 4]. For instance, since 2019, the annual increase in the NCBI Bacterial Assembly Database has exceeded the total volume of data accumulated during the previous two decades (Figure 1.1a). This expansion of genome collections has created a widening gap between the volume of sequenced data and storage capacities [5, 6]. Although computational resources improve over time, the growth of their capacities lags behind the pace of data accumulation. Consequently, the proportion of bacterial genomes that can be effectively searched and analyzed using traditional tools, e.g., BLAST [7] and its successors, has been decreasing exponentially (Figure 1.1b). These tools that used to be central to comparative genomics are becoming increasingly impractical for large-scale analyses, leaving a growing portion of existing genomic data effectively unavailable for routine research.



(a) Sequenced bacterial data grow exponentially. Over the last decade, the amount of newly sequenced genomic data has grown by several orders of magnitude. The graph shows the volume of assemblies deposited over the course of the last decade in the NCBI Bacterial Assembly Database, representing one of the most comprehensive databases in the field of bacterial genome assemblies. Image taken from [8].



(b) The amount of searchable bacterial data decreases exponentially. Since traditional search tools fail to keep up with the growth of sequenced data, the proportion of searchable data decreases exponentially. The plot shows the ratio of the data searchable by the BLAST Nucleotide Database and the size of the NCBI Bacterial Assembly database over the last two decades. Image taken from [9].

Figure 1.1: Illustrations of the exponential growth of bacterial data.

Since effective search in large collections requires identifying redundancies, substantial effort has been devoted to developing efficient compression techniques. Compression is a widely studied area in computer science, encompassing a broad range of techniques. General-purpose methods include dictionary-based compressors (e.g., `gzip, bzip2, xz`), statistical and probabilistic models (e.g., PPM, arithmetic coding), and more recent neural-network-based compressors that learn complex sequence regularities. Within this broad area, bacterial genomic data compression forms a specialized subfield that adapts these principles to the structure and redun-

dancy of bacterial sequences. Specialized tools, such as MBGC [10] and AGC [11] exploit extensive substring repetition across bacterial genomes to achieve substantial size reductions. An expanding class of k-mer-based methods has emerged, including large-scale indexing frameworks such as Metagraph [12], Themisto [13], and Fulgor [14]. These systems rely on advanced techniques including succinct k-mer dictionaries, minimizer-based sampling, and compressed graph representations to store and query massive bacterial genome collections. Despite their significant contributions, these approaches are highly optimized for specific queries rather than providing unified strategy, which motivates more systematic approach.

The recently proposed framework of phylogenetic compression [9] offers a promising strategy for scaling the storage of large-scale genomic collections. By reorganizing genomes into phylogenetically coherent batches and ordering them according to inferred evolutionary trees, phylogenetic compression localizes redundancy that would otherwise be dispersed across the dataset. When applied to large bacterial genome collections, it reduces storage requirements by one to two orders of magnitude. Phylogenetic compression is currently used as the central compression technique in large-scale resources such as the AllTheBacteria collection [4] and represents one of the most effective known methods for scaling genomic storage and search to millions of related bacterial genomes.

However, the theoretical foundations of phylogenetic compression remain little understood. Several fundamental questions are still open: How specifically does phylogenetic reordering lead to such dramatic improvements in compression and search efficiency? How can simple heuristics achieve orders-of-magnitude scaled improvements, despite the fact that even simplified formulations of compression problems are typically NP-hard? The gap between the empirical success and theoretical understanding suggests that the structure of bacterial genome collections may possess properties that make them particularly amenable to tree-guided optimization, but these properties have not been mathematically characterized.

In this thesis, we develop a formal framework for analyzing the compression capabilities of phylogenetic compression. We begin by modeling the structure of bacterial genome collections using the classical Infinite Sites Model (ISM), which provides a simple yet biologically meaningful description of how genomic variation accumulates along an evolutionary tree. We show that genome collections represented as binary matrices exhibit invariant structural properties across representations. We then formalize the task of compressing binary matrices as an optimization problem and show that in general, the problem is NP-hard. We further show that when the data arise from the ISM, the underlying evolutionary structure imposes constraints that make the problem tractable. In particular, we prove that phylogenetic reordering guided by an evolutionary tree reconstructed by the famous Neighbor-Joining algorithm [15] yields optimal compression in polynomial time under this model. Finally, we validate our theoretical predictions experimentally and show that, despite the simplifying assumptions, the model closely matches empirical behavior on real bacterial genome collections, deviating only minimally from optimal solutions provided by an exact solver.

Chapter 2

Background

2.1 DNA and DNA sequencing

DNA (Deoxyribonucleic Acid) is a fundamental molecule that carries the genetic instructions for growth, development, functioning, and reproduction of living organisms. It consists of two complementary linear strands of nucleotide units twisted into a double helix. Each nucleotide contains one of four nitrogenous bases: adenine (A), cytosine (C), guanine (G), and thymine (T). These bases pair specifically within the double helix structure: A pairs with T, and C pairs with G. The sequences encode biological information in an alphabet {`A, C, G, T`}.

The *genome* of an organism is the complete set of its DNA content. Since DNA is double-stranded, each molecule contains two complementary sequences whose orientation is determined by the chemical polarity of the sugar–phosphate backbone. The human genome consists of roughly 3.2 billion characters. In bacteria, the genomic material typically consists of a single circular chromosome ranging from several hundred thousand to a few million characters, together with additional extrachromosomal elements such as plasmids or other genetic elements.

Genomes are studied through a process called *DNA sequencing*, in which the DNA molecule is fragmented into smaller pieces, and the sequences of these fragments are then called *reads*. Sequencing technologies differ substantially in throughput, accuracy, and read length, and can be grouped into three generations [16, 17]. First-generation sequencing, represented primarily by Sanger sequencing, produces highly accurate reads of roughly 700–1,000 characters but offers only limited throughput, typically on the order of a few million characters per run. Second-generation technologies, such as Illumina platforms [18], increased throughput by several orders of magnitude by sequencing millions of short fragments in parallel at the cost of shorter read lengths (typically 50–300 characters). Third-generation sequencing technologies, including Oxford Nanopore [19] and PacBio [20] single-molecule platforms, enable much longer reads (often thousands to tens of thousands of characters), though with higher error rates than earlier generations. The genome is then computationally assembled from these reads, most commonly using assembly algorithms based on de Bruijn or overlap graphs.

In the context of this thesis, we focus exclusively on bacterial genomes and model each genome as a linear string over the alphabet {`A, C, G, T`}. This mathematical abstraction assumes that the chromosome has been linearized and that the resulting sequence is complete and free of errors. Moreover, we restrict our analysis to chromosomal DNA, deliberately excluding plasmids.

2.2 Mathematical representation of bacterial genome collections

2.2.1 Strings and k-mers

An *alphabet* is a finite, non-empty set of symbols called *letters*. We let $|S| = L$ denote the length of the string $S = S_1 S_2 \cdots S_L$. A *genome* is a finite string $G = G_1 G_2 \cdots G_L$ over the alphabet {`A, C, G, T`}. A contiguous substring of length k in a genome G is called a *k-mer*. A prefix of G of length m is denoted $G[1 \cdots m]$.

Example 1. Consider the genome $G =$ `CAAGCT`. The string `AGCT` is a 4-mer of the genome G. The 3-mers of this genome are `CAA`, `AAG`, `AGC`, and `GCT`. The string `CACT` is not a k-mer of genome G since it is not its contiguous substring. The Hamming distance of genomes `CAAGCT` and `CATACG` is 3.

Given a k-mer k_i, its *reverse complement*, denoted $RC(k_i)$, is the k-mer obtained from k_i by applying the following two operations: (i) reverse the order of letters, and (ii) replace each character with its complementary character, i.e., `A` is substituted with `T` and vice versa, and `G` is substituted with `C` and vice versa. The lexicographically smaller of k_i and $RC(k_i)$ is called *canonical form of* k_i.

There are two equivalence models for sets of k-mers. In the *unidirectional* model, each k-mer is equivalent only to itself. In the *bidirectional* model, a k-mer is treated as equivalent to its reverse complement. The bidirectional model more accurately reflects the nature of sequencing data, since DNA is double-stranded and the sequenced fragment cannot be distinguished from its reverse complement. In the parts of our theoretical framework that operate on k-mer-based representations, we use only canonical k-mers; for each equivalence class containing a k-mer and its reverse complement, the canonical k-mer serves as the representative. Unless stated otherwise, we refer to the equivalence classes represented by canonical k-mers as k-mers.

Example 2. Consider the genome $G =$ `TAAGCT` and its 5-mer `TAAGC`. Its reverse complement is $RC($`TAAGC`$) =$ `GCTTA`. The canonical 5-mers of G are `GCTTA` and `AAGCT`.

2.2.2 De Bruijn graphs, unitigs, Bloom filters

Definition 1 (De Bruijn graph)**.** Let $C = \{G_1, \ldots, G_n\}$ be a collection of genomes over the DNA alphabet $\Sigma = \{A, C, G, T\}$. For a fixed integer $k \geq 1$, the *de Bruijn graph of order* k constructed from the collection C, denoted $DB_k(C) = (V, E)$, is a directed graph defined as follows. The vertex set

$$V = \{u \in \Sigma^k \mid u \text{ occurs as a substring of some genome in } C\}$$

consists of all k-mers present in the collection. For every $(k+1)$-mer $w \in \Sigma^{k+1}$ occurring in some genome G_i, let $\text{pref}(w) = w[1 \cdots k]$ and $\text{suff}(w) = w[2 \cdots k+1]$. The graph contains a directed edge

$$\text{pref}(w) \to \text{suff}(w)\,.$$

Parallel edges of the same $(k+1)$-mer are represented by a single edge.

A *colored de Bruijn graph* extends this definition by assigning each genome in the input collection a distinct color and annotating each node with the set of colors of the genomes in which the associated k-mer occurs. The non-branching paths whose nodes share the same set of colors can be collapsed into single nodes that spell the corresponding substrings, called *unitigs*.

Definition 2 (unitig)**.** Let C be a collection of genomes and for a fixed integer $k \geq 1$, let $DB_k(C) = (V, E)$ be the colored de Bruijn graph of order k of C, where each vertex $u \in V$ is annotated with a color set $\text{col}(u)$. Let p be a directed path $u_1 \to u_2 \to \cdots \to u_\ell$ in $DB_k(C)$ such that

1. for every internal vertex u_j, $1 < j < \ell$, we have $\deg^-(u_j) = \deg^+(u_j) = 1$, where $\deg^+$ denotes the number of outgoing edges and $\deg^-$ denotes the number of incoming edges,
2. all vertices on the path share the same color set, i.e.,

$$\text{col}(u_1) = \text{col}(u_2) = \cdots = \text{col}(u_\ell)\,.$$

3. either $\ell > 1$ and at least one of the endpoints satisfies

$$\deg^-(u_1) \neq 1 \text{ or } \deg^+(u_1) \neq 1 \text{ or } u_1 \text{ does not share the same color set as some of its neighbors}\,,$$

   or similarly for u_ℓ, or p is a simple directed cycle on nodes of the same color set in which every vertex satisfies $\deg^- = \deg^+ = 1$,
4. the path is maximal with respect to the degree and color-consistency conditions, i.e., it cannot be extended on either end while preserving all conditions above.

The string spelled by the path p, obtained by taking the k-mer u_1 and appending the last character of each successive vertex $u_2, \ldots, u_\ell$, is called *unitig*.

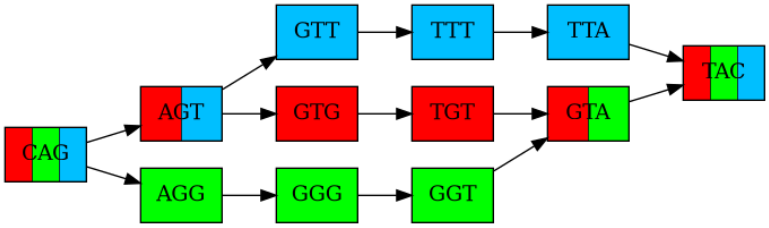


Figure 2.1: Example of a de Bruijn graph. Collection $C = \{\texttt{CAGTGTAC}, \texttt{CAGGGTAC}, \texttt{CAGTTTAC}\}$ corresponds to the following de Bruijn graph of order $k = 3$, with genomes associated with colors red, green, and blue, respectively. The unitigs in this graph are `CAG`, `AGT`, `GTTTA`, `GTGT`, `AGGGT`, `GTA`, `TAC`.

A Bloom filter is a probabilistic data structure that indicates whether an element is a member of a set. Bloom filter is built using hash functions, which allows false positives, but not false negatives. The false positive probability depends on the parameters of the Bloom filter. We will view Bloom filter as a binary representation of k-mer set that represents given genome.

Definition 3 (Bloom filter)**.** Let G be a genome over the alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$, fix $k \in \mathbb{N}$, and let $K_G = \{k_1, \ldots, k_n\}$ be the set of its distinct k-mers. A *Bloom filter* of the k-mer set of the genome G is a probabilistic data structure defined by

- a collection of μ independent hash functions $h_1, \ldots, h_\mu : \Sigma^k \to \{1, \ldots, m\}$,
- a bit array $B_G \in \{0,1\}^m$ of length $m \geq 1$, initially all zeros, which for each k-mer $k_i \in K_G$ sets $B_G[h_1(k_i)] = B_G[h_2(k_i)] = \cdots = B_G[h_\mu(k_i)] = 1$.

To query whether a given k_j belongs to K_G, the Bloom filter returns true if and only if $B_G[h_1(k_j)] = B_G[h_2(k_j)] = \cdots = B_G[h_\mu(k_j)] = 1$.

2.2.3 Binary matrix representations

In this thesis, we will mainly focus on binary matrices representing genome collections. These matrices can be built from genomes themselves or from standard bioinformatics data structures introduced above.

Definition 4 (SNP matrix)**.** Let $C = \{G_1, \ldots, G_n\}$ be a collection of genomes of equal length L and X be a genome of the same length. The SNP matrix $S \in \{0,1\}^{L \times n}$ of the collection C with respect to X as a reference genome is defined as

$$S_{ij} = \begin{cases} 1 & \text{if } G_j[i] \neq X[i], \\ 0 & \text{otherwise.} \end{cases}$$

Definition 5 (k-mer matrix)**.** Let $\{G_1, \ldots, G_n\}$ be a collection of genomes, and fix $k \in \mathbb{N}$. Define $\{k_1, \ldots, k_D\}$ as the set of all distinct k-mers that appear in at least one genome of the collection. The k-mer matrix $K \in \{0,1\}^{D \times n}$ is defined as

$$K_{ij} = \begin{cases} 1 & \text{if } k_i \text{ occurs in } G_j, \\ 0 & \text{otherwise.} \end{cases}$$

Example 3. Consider a genome collection $C = \{CAGTGTAC, CAGGGTAC, CAGTTTAC\}$ and $k = 3$, and for clarity consider unidirectional model. The k-mer matrix of C is

$$\begin{array}{c} \\ GTT \\ TTT \\ TTA \\ CAG \\ AGT \\ GTG \\ TGT \\ GTA \\ TAC \\ AGG \\ GGG \\ GGT \end{array} \begin{array}{c} \begin{array}{ccc} G_1 & G_2 & G_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Definition 6 (unitig matrix)**.** Let $\{G_1, \ldots, G_n\}$ be a collection of genomes, fix $k \in \mathbb{N}$, and F be the number of its distinct unitigs $u_1, \ldots, u_F$ in the corresponding de Bruijn graph $DB_k(C)$. The unitig matrix $U \in \{0,1\}^{F \times n}$ is defined as

$$U_{ij} = \begin{cases} 1 & \text{if } u_i \text{ is present in } G_j, \\ 0 & \text{otherwise.} \end{cases}$$

Example 4. Consider the collection $C = \{\texttt{CAGTGTAC}, \texttt{CAGGGTAC}, \texttt{CAGTTTAC}\}$ used in Figure 2.1 and $k = 3$. Its unitig matrix is

$$\begin{array}{c} \\ CAG \\ AGT \\ GTTTA \\ GTGT \\ AGGGT \\ GTA \\ TAC \end{array} \begin{array}{c} \begin{array}{ccc} G_1 & G_2 & G_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \end{array}.$$

Bloom filters can also be organized into matrix-like structures. Here, we adopt the formulation introduced in [21].

Definition 7 (Bloom-filter matrix)**.** Let $C = \{G_1, \ldots, G_n\}$ be a collection of genomes, fix $k \in \mathbb{N}$, and let $\{k_1, \ldots, k_D\}$ be the set of all distinct k-mers that appear in at least one genome of the collection C. Let m be the length of a Bloom filter array and μ be the number of independent hash functions $h_1, \ldots, h_\mu : \Sigma^k \to\to \{1, \ldots, m\}$. Then the *Bloom-filter matrix* $B \in \{0,1\}^{m \times n}$ is defined as

$$B_{ij} = \begin{cases} 1 & \text{if there exists a } k\text{-mer } k_s \text{ in } G_j \text{ for which } i \in \{h_1(k_s), \ldots, h_\mu(k_s)\}, \\ 0 & \text{otherwise.} \end{cases}$$

Equivalently, the columns of the matrix correspond to Bloom filter bit arrays of the individual genomes, i.e., $B_{\bullet j} = B_{G_j}$ using the notation from Definition 3.

2.3 Selected algorithmic tools and concepts

2.3.1 Distance measures

Many constructions in this thesis rely on specific distance measure. We will frequently use the *Hamming distance* between two strings of equal length, defined as the number of positions at which their characters differ. In addition, we make use of two important classes of tree-based distance metrics introduced below.

Definition 8 (Additive distances)**.** Let Y be a finite set of objects. An *additive distance* is a metric $d : Y \times Y \to \mathbb{R}_0^+$ for which there exists a weighted tree T with leaf set Y such that for any $x, y \in Y$,

$$d(x,y) = \sum_{e \in P(x,y)} w(e),$$

where $P(x,y)$ is the unique path between x and y in T, and $w(e)$ is the weight of edge e.

Definition 9 (Ultrametric distances)**.** Let Y be a finite set of objects. A metric $d : Y \times Y \to \mathbb{R}_0^+$ is *ultrametric* if for all $x, y, z \in Y$,

$$d(x,y) \leq \max\{d(x,z), d(z,y)\}.$$

Equivalently, all leaves are equidistant from the root in the corresponding tree.

2.3.2 Phylogenetic inference

Phylogenetic inference is the computational reconstruction of evolutionary relationships among organisms from their genomic sequences. It plays a central role in bioinformatics, as understanding how genomes are related through common ancestry is essential for tasks such as genome annotation, comparative genomics, outbreak

tracking, and the study of molecular evolution. Phylogenetic methods provide a principled framework for interpreting sequence similarity not merely as a measure of distance, but as evidence of descent with modification. Evolutionary relationships can be represented using various mathematical structures, in this work, we focus exclusively on tree-based models.

A *phylogenetic tree* is a mathematical structure representing evolutionary relationships among a set of taxa. Formally, it is a rooted or unrooted connected acyclic graph $T = (V, E)$ whose leaves correspond to observed genomes and whose internal nodes represent hypothetical ancestors. Edges can be weighted by genetic distance or substitution rates. The topology of the tree captures the branching order of descent, while edge weights quantify evolutionary time or accumulated change. Importantly, there is no single "correct" biological tree: the inferred structure depends on the chosen method, evolutionary model, and data quality. In this thesis, we focus on the phylogenetic inference methods that are directly used in our theoretical model or evaluation pipeline.

Several methodological categories of phylogenetic inference exist. *Distance-based* approaches (e.g., Neighbor Joining or UPGMA) construct trees from pairwise sequence distances. *Character-based* approaches, such as maximum likelihood, evaluate explicit probabilistic models of sequence evolution. *Bayesian* methods incorporate prior information and use sampling to estimate posterior distributions over trees. Each class of methods balances computational tractability with biological realism, and each provides different insights into the evolutionary history encoded in genomic data.

Selected computational methods for phylogenetic inference

Neighbor Joining (NJ) Neighbor Joining [15] is a widely used distance-based phylogenetic method, i.e., it constructs a tree solely from pairwise distances between genomes, without requiring the sequences themselves or prescribing how those distances must be defined. Given a set of objects and their pairwise distances, NJ produces a tree that exactly represents those distances by weighted branches whenever they are additive (Definition 8). For such distances, the underlying tree T is uniquely determined by d (up to branch rotations), and NJ reconstructs exactly this unrooted tree in $\mathcal{O}(n^3)$ time.

In the context of this thesis, NJ is applied to genome distances estimated from k-mer based sketches [22], an approach that allows efficient estimation of pairwise distances. Given the relevance of the theoretical properties of NJ to our results, their overview is presented in Appendix A.

Unweighted pair group method with arithmetic mean (UPGMA) UPGMA is another classical distance-based algorithm. It assumes an even stronger condition of so-called *molecular clock*, meaning that all genomes evolve at a constant rate [23]. Under this assumption the distances between genomes must satisfy the ultrametric property (Definition 9). For such distances, UPGMA reconstructs the correct rooted tree. The algorithm works in $\mathcal{O}(n^2)$ time and proceeds by iteratively clustering the closest pair of genomes or clusters, updating the distance matrix using arithmetic means, and continuing until a single rooted tree remains. In this thesis, UPGMA is combined with sketch-based distance estimation for scalability.

Randomized accelerated maximum likelihood (RAxML) RAxML [24] is a widely used maximum-likelihood (ML) method that aims to identify the tree topology and branch lengths that maximize the probability of observing the given sequence data under a specified evolutionary model. As an ML framework, it incorporates mutation processes, substitution rates, and other biological factors, enabling statistically robust phylogenetic inference. RAxML supports a broad range of substitution models and is optimized for larger datasets than previous ML methods. In this thesis, we use a RAxML tree derived from a recombination-filtered core-genome alignment [25], downloaded from public repository.

2.3.3 Phylogenetic compression

Phylogenetic compression [9] is a high-level compression method that exploits evolutionary relationships among genomes to group closely related sequences together. Because genomes are not independent strings of characters but share a common evolutionary history, they contain substantial redundancy. Phylogenetic compression systematically leverages this redundancy to achieve substantially better compression than generic algorithms. Its core idea is to organize genomes according to their phylogenetic tree and compress them in a way that respects this evolutionary structure.

More specifically, phylogenetic compression involves four steps.

1. **Clustering and batching:** The collection is divided into phylogenetically related groups (e.g., one species per group) which are then batched into groups of balanced size.

2. **Phylogenetic inference:** Within each batch, the method infers a phylogeny. The phylogenetic tree then serves as a guide for the next step.
3. **Data reordering:** Genomes are reordered left-to-right according to the inferred phylogeny.
4. **Data compression:** The reordered dataset is compressed by a low-level compressor.

The method is highly versatile and can be adapted to different data representations through so-called protocols. The four core steps are very general and must be instantiated and tailored to a specific setting before they can be applied in practice. For instance, a protocol for a collection of genome assemblies might involve clustering them by species to batches of similar sizes, reordering each batch according to a phylogeny inferred with `mashtree` [26], and then compressing the reordered genomes using `xz`. In our theoretical model, we use a simplified protocol that defines how phylogenetic compression is performed and specifies the tools used at each step.

In practice, phylogenetic compression improves compression ratios of assemblies, de Bruijn graphs, and k-mer indices by one to two orders of magnitude compared to conventional techniques. It enhances the performance of traditional compression tools such as *xz* as well as specialized tools such as *MBGC* [10]. It is especially effective for large genomic datasets comprising of both closely related genomes as well as highly diverse ones.

2.3.4 Run-length encoding (RLE)

Run-length encoding (RLE) is one of the simplest and oldest forms of lossless data compression. Its core idea is to exploit consecutive repetitions of the same symbol, called *runs*, by representing them in a compact form. Instead of storing each symbol individually, RLE stores a pair (s, ℓ), where s is the repeated symbol and ℓ is the count of its occurrences (i.e., length of the run).

Example 5. The string $CAAGTTTT$ can be encoded in RLE as $(C, 1), (A, 2), (G, 1), (T, 4)$.

RLE reduces storage requirements by collapsing long regions of equal characters. Its encoding and decoding procedures are extremely fast and require minimal memory, but RLE is beneficial only when sufficiently long runs are present; on highly variable data, the compressed form may not be smaller than the original. For this reason, RLE is often combined with other compression or indexing techniques in large-scale applications. It appears across domains where data naturally contains long uniform segments [27]: classic examples include early image and multimedia formats, such as bitmap graphics and fax transmission standards, which exploit RLE to compress large regions of identical color or intensity. In text and binary compression, RLE is frequently used as a lightweight preprocessing step to expose structure for more advanced algorithms.

In the context of this thesis, RLE serves as the low-level compressor of binary matrices in our simplified phylogenetic compression protocol. The compressed matrix size is measured by the total number of runs across all rows, providing a simple metric for evaluating binary matrix compression.

2.3.5 Traveling Salesperson Problem (TSP)

The Traveling Salesperson Problem (TSP) is a classical problem in combinatorial optimization and theoretical computer science. Informally, it asks for the shortest possible route that visits each city in a given set exactly once and returns to the starting point. Formally, given a set of n vertices and their pairwise distances (i.e., their complete weighted graph), the goal is to find a minimum-weight cycle that passes through all vertices. An equal formulation is to find a minimum-weight Hamiltonian cycle in a complete graph. A *Hamiltonian cycle* is a cycle that visits every vertex exactly once; however, the classical Hamiltonian cycle problem is defined for arbitrary (not necessarily complete or weighted) graphs and concerns only the existence of such a cycle, not its length. TSP is one of the most extensively studied NP-hard problems and serves as a benchmark for algorithm design, approximation theory, and computational complexity.

Problem 1 (Traveling Salesperson Problem (TSP)).
Input: Complete weighted graph $G = (V, E)$ with $|V| = n$ vertices and a distance function $d : V \times V \to \mathbb{R}_0^+$
Output: Permutation of vertices π minimizing $d(v_{\pi_1}, v_{\pi_2}) + d(v_{\pi_2}, v_{\pi_3}) + \ldots + d(v_{\pi_{n-1}}, v_{\pi_n}) + d(v_{\pi_n}, v_{\pi_1})$

The computational complexity of TSP depends strongly on the structure of the distance function. For general distances, TSP is NP-hard, and the decision version (asking whether a tour of length at most L exists) is NP-complete [28]. A brute-force search would require evaluating $(n-1)!/2$ possible tours. When the distances

form a metric, i.e., they are symmetric and satisfy the triangle inequality, the problem remains NP-hard, but admits polynomial-time approximation algorithms [29]. A classical example is the Christofides–Serdyukov algorithm [30], which guarantees a tour of length at most 1.5 times the optimum. Even stronger approximation schemes exist for Euclidean TSP, although the problem is still NP-hard [31], and similar holds for Hamming distances [32]. The situation changes dramatically when the distances are *additive*, meaning they arise from path lengths on a tree graph. In this case, the TSP becomes polynomial-time solvable.

In our work, genomes play the role of "cities" and pairwise distances are defined by the Hamming distances between their binary encodings in genomic matrices. The TSP formalizes the problem of finding an optimal ordering of genomes.

Chapter 3

Mathematical framework for phylogenetic compression modeling

The goal of this chapter is to formalize a simplified protocol of phylogenetic compression, and formalize compression within this specific framework as an optimization problem. Doing so enables direct comparison between the performance of phylogenetically guided compression, an optimal solution, and a standard compression protocol without evolutionary guidance. This framework also allows us to contrast the theoretical behavior of the model on idealized, evolutionarily structured data with its behavior on arbitrary data without such assumptions.

3.1 Overview of modeling methodology

To construct a mathematical model capable of explaining the performance of phylogenetic compression, we must specify the exact compression protocol. This requires several key choices: selecting the type of bacterial genomic data and the evolutionary assumptions we impose on it; defining an appropriate data representation and analyzing its properties under those assumptions; and specifying the concrete components of the phylogenetic compression pipeline, including the clustering strategy, the algorithm used to infer the phylogenetic tree, and the low-level compressor. With these elements in place, we can formally express compression as an optimization problem and systematically compare phylogenetically informed solutions with both optimal and unadjusted baselines.

3.2 Modeling structure of data by Infinite sites model (ISM)

When studying the structure of genomic sequences from mathematical perspective, the essential observation is that genomes are not arbitrary strings but products of an underlying evolutionary process. Their modeling therefore requires abstractions that capture both the discrete nature of the genomic sequences and the constraints imposed by evolutionary inheritance and mutations. In practice, this means that mathematical models have to balance biological plausibility with mathematical tractability. By grounding our compression protocol in such models, we can explicitly incorporate evolutionary assumptions into the data representation and study its properties.

3.2.1 Perfect phylogeny and Infinite sites model

To model a simplified evolutionary process that shapes the structure of data [6, 9], we adopt the classical *perfect phylogeny model* [33] (Figure 3.1a). It describes evolutionary relationships between given objects (e.g., species or other taxa) and their features that are either present or absent in the objects. Perfect phylogeny models the evolutionary history of these objects and their features under the assumption that each feature appears only once and is never lost or reverted. This model has been widely studied in the context of graph and complexity theory, reconstruction algorithms and computational biology [34, 35, 36, 37, 38].

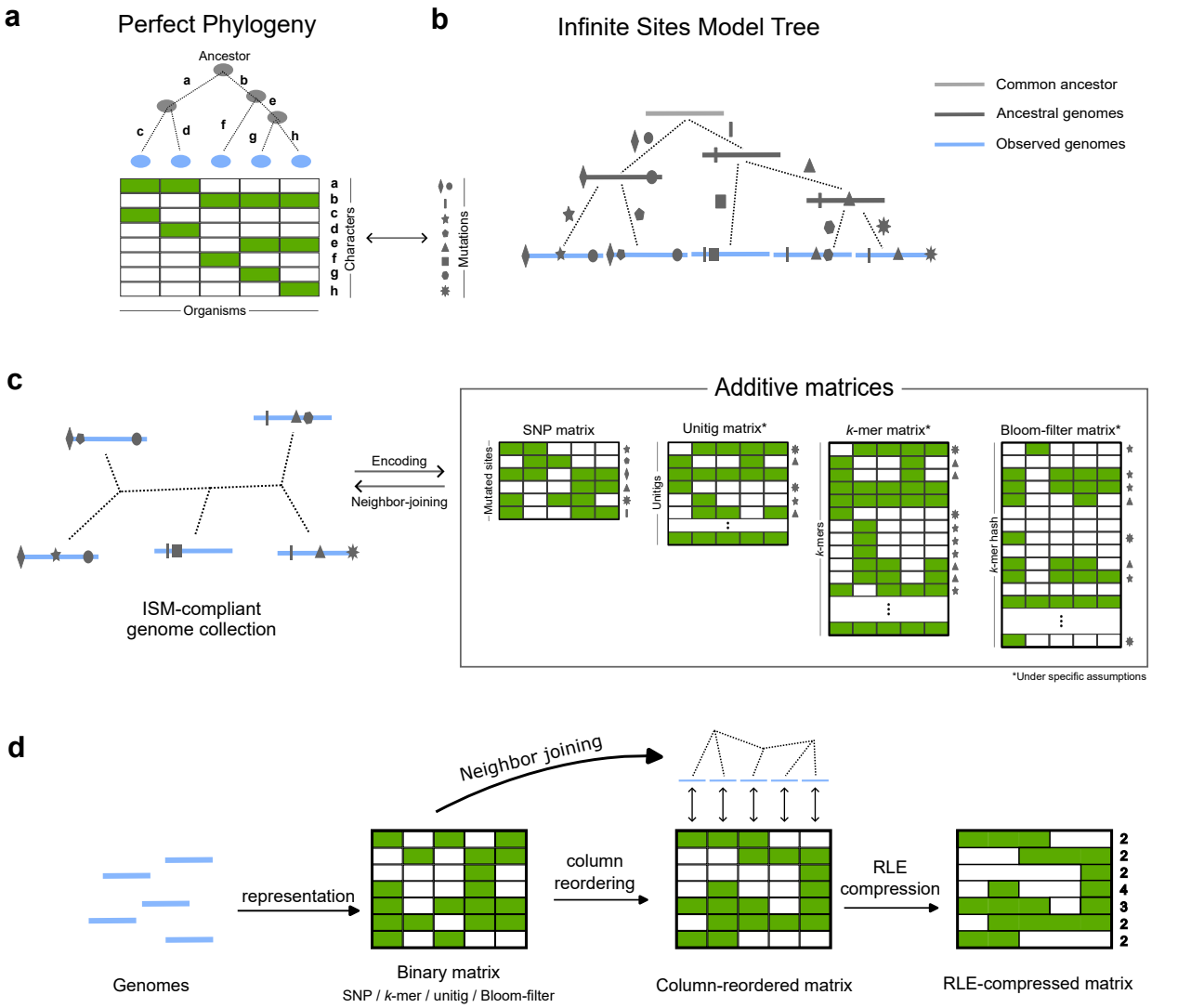


Figure 3.1: Overview of our modeling methodology. a) Perfect phylogeny: a tree-based model of feature emergence, which can be characterized by a perfect phylogeny matrix. **b)** The Infinite Sites Model (ISM): the ISM instantiates a perfect phylogeny for genomes and point mutations, producing a rooted binary tree with observed genomes as leaves. **c)** Binary matrices representing genome collections: ISM-compliant data define an additive distance, and SNP, unitig, k-mer and Bloom-filter matrices derived from this data all induce additive Hamming distances on their columns, some of them under additional assumptions. The underlying unrooted tree can be recovered from these matrices via Neighbor Joining, even if the evolutionary history is unknown. **d)** a simplified protocol of phylogenetic compression: on ISM-compliant matrices, phylogeny-guided RLE compression performs optimally.

A perfect phylogeny is a rooted tree T where each object is associated with exactly one leaf of T and each feature corresponds to exactly one edge of the tree. Each object contains exactly those features that are associated with the edges marking the unique path from the root of the tree to the corresponding leaf. In other words, once a feature appears in some ancestral object, it is then present in all the descendants of that object and not anywhere else in the tree.

Perfect phylogeny can be represented as a binary matrix M whose rows correspond to features and columns correspond to the objects. The value $M_{ij} = 1$ indicates that feature i is present in object j. Such a matrix is called *perfect phylogeny matrix*. A central result for our work is the necessary and sufficient condition for a given binary matrix to be a perfect phylogeny matrix.

Lemma 1 ([37], [39])**.** Let $M \in \{0,1\}^{m \times n}$ be a binary matrix. For each row i, let S_i be the set of column indices where row i contains a 1, i.e., $S_i = \{j \mid M_{ij} = 1\}$. Then M is a perfect phylogeny matrix if and only if for every pair of rows i, j, the sets S_i and S_j are either disjoint or one contains the other.

At the level of DNA sequences, perfect phylogenies naturally arise under the *Infinite Sites Model* (ISM) [40] (Figure 3.1b). This simplified yet highly tractable model represents genomes as long strings, and the assumption of each feature appearing exactly once and never reverting translates to the ISM as single-site mutations in the genomes' novel positions. Each site in the genome can mutate at most once in evolutionary history, and once mutated, the derived state is preserved in all descendant lineages. More specifically, the ISM assumes that (1) each new mutation arises at a unique genomic position (no recurrence or reversal), and (2) there is no recombination, i.e., all genomic transfer is purely vertical in the tree. The ISM underpins a wide range of population-genetic methods and tools, including studies of neutral evolution [41], [42], demographic inference [43], and simulation frameworks such as the Wright–Fisher model [40], [44], Moran model [45], [46], and coalescent models [47], [48]. Efficient simulators supporting the ISM include fwdpy11 [49], fastsimcoal2 [50], and SLiM 3 [51].

While the ISM remains a cornerstone for modeling genetic drift and mutation dynamics, our interest lies in the combinatorial patterns it guarantees; we formalize this combinatorial structure as an *ISM tree*.

Definition 10 (ISM tree)**.** Let $C = \{G_1, \ldots, G_n\}$ be a genome collection composed of a set of equal length genomes. We call a rooted binary tree $\mathcal{T}$ an *ISM tree* of the collection if

1. The root of $\mathcal{T}$ contains no mutated positions,
2. the leaves of $\mathcal{T}$ are exactly $\{G_1, \ldots, G_n\}$,
3. every edge is labeled by a set of mutations at unique positions, i.e., one position never appears in more than one edge label,
4. each descendant inherits the mutations accumulated on all edges from root to that node,
5. edge weights are derived as the number of mutations in the corresponding edge label.

We call such a collection C *ISM-compliant*.

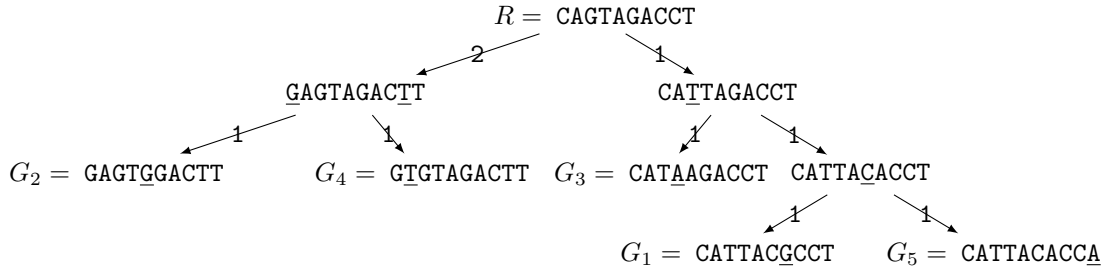


Figure 3.2: Example of an ISM tree. Here, the ISM-compliant collection C contains genomes $G_1, \ldots, G_5$, and the genome R is the common ancestor. For clarity, we list the edge weights explicitly and novel mutations are highlighted in genomes as underlined positions.

3.2.2 Modeling genome collections with ISM-compliant matrices

As the complete evolutionary history is rarely known, practical applications require representations that do not rely on the access to the true ancestral genome. For instance, given a set of related genomes, such as genomes from a hospital outbreak of *Staphylococcus aureus* [52], a common approach is to select a reference genome X from within or near the dataset, and then encode all other genomes as binary vectors relative to X, which effectively yields a SNP matrix (Definition 4). Other widely used matrix representations, e.g., k-mer and unitig matrices or Bloom-filter-based representations, also represent each genome as a binary vector over a set of attributes. We formalize this through the concept of ISM-compliant matrices, unifying these matrix types used throughout bioinformatics. Here, columns represent observed genomes and rows correspond to individual attributes, and we restrict our attention to binary matrices.

Definition 11 (ISM-compliant matrix)**.** Let $M \in \{0,1\}^{m \times n}$ be a binary matrix. For each row i, define two subsets of column indices $T_i = \{j \mid M_{ij} = 1\}$ and $F_i = \{j \mid M_{ij} = 0\}$. Then M is called *ISM-compliant* if for each pair of rows r, s, at least one of these four intersections is empty $T_r \cap T_s$, $T_r \cap F_s$, $F_r \cap F_s$, $F_r \cap T_s$.

Equivalently, M is ISM-compliant if no pair of rows contains all four binary patterns (00, 01, 10, 11). This is called the *four-gamete condition*, used for recombination detection [53] and perfect-phylogeny algorithms [54]. The complexity of naive verification of ISM-compliance for a given binary matrix $M \in \{0,1\}^{m\times n}$ requires $\mathcal{O}(nm^2)$ time, but specialized data structures allow improvements down to $\mathcal{O}(nm)$ [33].

ISM-compliant matrices strictly generalize perfect phylogeny matrices. When we view the feature emergence in perfect phylogeny as a single-site mutations in the ISM tree, the SNP matrix with the root of the tree as a reference yields exactly a perfect phylogeny matrix. Conversely, ISM-compliant matrices relax the constraint of evolutionary directionality: the referenced features are no longer required to be those of the root, but are compared only among the observed genomes.

Lemma 2. Let $\mathcal{M}_{PP}$ denote the class of perfect phylogeny matrices, and let $\mathcal{M}_{ISM}$ denote the class of ISM-compliant binary matrices. Then $\mathcal{M}_{PP} \subsetneq \mathcal{M}_{ISM}$.

Proof. Let us use the notation of a given binary matrix $M \in \{0,1\}^{m\times n}$ with the sets $T_k = \{j \mid M_{kj} = 1\}$ and $F_k = \{j \mid M_{kj} = 0\}$. Then $M \in \mathcal{M}_{PP}$ if for every pair of rows i, j, either the T sets are disjoint ($T_i \cap T_j = \emptyset$) or one is a subset of the other ($T_i \cap F_j = \emptyset$ or $T_j \cap F_i = \emptyset$). Clearly, this satisfies the definition of ISM-compliance. Furthermore, the set of matrices that do not comply with this condition but satisfy for every pair of rows $F_i \cap F_j = \emptyset$ is ISM-compliant but not a perfect phylogeny matrix. □

Example 6. Consider the following binary matrix M representing 3 genomes, and its corresponding sets of indices.

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \qquad \begin{aligned} T_1 &= \{1,3\} & F_1 &= \{2\} \\ T_2 &= \{2\} & F_2 &= \{1,3\} \\ T_3 &= \{1,2\} & F_3 &= \{3\} \\ T_4 &= \{1,3\} & F_4 &= \{2\} \end{aligned}$$

It is ISM-compliant, but not a perfect phylogeny matrix, because T_1 and T_3 are neither subsets of one another nor disjoint.

A key property of ISM-compliant matrices is that pairwise Hamming distances between their columns induce an additive metric (Definition 8). For perfect phylogeny matrices, this is trivial: each feature corresponds to exactly one edge, so the Hamming distance equals the total number of features corresponding to the edges along the path between leaves. ISM-compliant matrices inherit this property due to the four-gamete condition.

Theorem 1. Hamming distances induced by the columns of an ISM-compliant matrix are additive.

Proof. Recall the notation from the definition of an ISM-compliant matrix: $M \in \{0,1\}^{m\times n}$, $T_i = \{j \mid M_{ij} = 1\}$ and $F_i = \{j \mid M_{ij} = 0\}$. M is ISM-compliant, so for each pair of its rows r, s, at least one of the intersections $T_r \cap T_s$, $T_r \cap F_s$, $F_r \cap T_s$, $F_r \cap F_s$ is empty. Equivalently, no pair of binary features given by rows r, s per column can produce all four pairs $(0,0), (0,1), (1,0), (1,1)$ (the four-gamete condition).

We want to show that the columns in M, corresponding to binary representations of genomes, can be organized as leaves of a binary tree. A classical result of Peter Buneman [55] shows exactly this; each row $r \in M$ induces a *split* (bipartition) (T_r, F_r) of the columns. By [55], if the splits satisfy the four-gamete condition, there is a unique binary tree whose edges realize exactly those splits and whose leaves correspond to the set of columns. In particular, each split (T_r, F_r) corresponds to a unique edge, whose removal separates the leaves into T_r and F_r.

The Hamming distance $d(x, y)$ between two columns $x, y \in M$ counts the number of rows in which their values differ. In the split-induced tree, the row r (which corresponds to exactly one edge e_r) separates x and y exactly when the path from x to y crosses e_r. Let us give weight $w(r)$ to the edge e_r if the split is duplicated $w(r)$ times in the rows of M. Then the Hamming distance $d(x, y)$ equals the length of the path between x and y in the tree and is therefore additive. □

3.2.3 Examples of common ISM-compliant matrices

When built on an ISM-compliant genome collection, many of frequently used representation matrices are ISM-compliant as well, although some need additional assumptions (Figure 3.1c). Here, we will show that SNP matrix is ISM-compliant under the condition of compatible reference. Furthermore, we show that k-mer, unitig, and Bloom-filter matrices are ISM-compliant under k-mer-based ISM tree and low collision rate of the Bloom filter.

SNP matrix

Definition 12 (Compatible reference genome)**.** Let $C = \{G_1, \ldots, G_n\}$ be a collection of equal-length genomes for which there exists an ISM tree $\mathcal{T}$ with C as its leaves. Let X be a genome such that for any position j, $X[j] \in \{G_i[j] \mid G_i \in C\}$, i.e., X does not introduce a novel nucleotide at any position. We call X a *compatible reference genome* of the collection C.

Example 7. Let us consider the ISM-compliant collection C from Figure 3.2. A compatible genome for this collection could be for example the root $R =$ `CAGTAGACCT` or other internal genome, one of the genomes from the collection $G_1 =$ `CATTACGCCT`, or a genome from outside the tree $G_c =$ `CAGTACGCTT`. The genome $G_n =$ `CATTACACTG` is not compatible with C since the last site in G_n contains a novel nucleotide G which is not present in any other genome in C.

Theorem 2. Let $C = \{G_1, \ldots, G_n\}$ be an ISM-compliant genome collection of equal length genomes with an ISM tree, and X be a genome compatible with this collection. Then the SNP matrix built on the collection C with the reference X is ISM-compliant.

Proof. Let us call the corresponding SNP matrix $S \in \{0,1\}^{L \times n}$, where L is the length of the genomes, and use the notation of the subsets of column indices $T_i = \{j \mid S_{ij} = 1\}$, $F_i = \{j \mid S_{ij} = 0\}$ for each row i in the SNP matrix S. We want to show that for each pair of rows r, s, at least one of the four intersections $T_r \cap T_s, T_r \cap F_s, F_r \cap T_s, F_r \cap F_s$ is empty.

Any row that is all-ones or all-zeros automatically satisfies this condition, so we restrict attention to the remaining (nontrivial) rows. Rows in an SNP matrix represent individual sites that mutate along the edges of the ISM tree $\mathcal{T}$, so each row is naturally associated with an edge of $\mathcal{T}$. This edge divides $\mathcal{T}$ into two subtrees, whose leaf sets correspond to the index sets T_r, F_r, T_s, F_s.

Let us choose two nontrivial rows r, s in S. The edge associated with r divides the tree into subtrees with leaf sets corresponding to T_r and F_r. There are four possible scenarios, based on whether the edge s lies within the subtree with leaves T_r or within the subtree with leaves F_r.

- If the edge associated with s lies within T_r, then either
  - $F_s \subseteq T_r$, therefore $F_s \cap F_r = \emptyset$,
  - $T_s \subseteq T_r$, therefore $T_s \cap F_r = \emptyset$.
- If it lies within F_r, then either
  - $F_s \subseteq F_r$, therefore $F_s \cap T_r = \emptyset$,
  - $T_s \subseteq F_r$, therefore $T_s \cap T_r = \emptyset$.

In all cases, at least one intersection must be empty, therefore the matrix S is ISM-compliant. □

k-mer matrix

Next, we will examine k-mer matrices, which represent genomes based on the presence or absence of k-mer. To preserve the fundamental ISM properties, we need to modify the assumptions we lay on the genome collection and translate the concept of individual positions to k-mer sets. Essentially, we need to restrict how closely individual mutations can appear within a genome given a fixed k since we want to prevent k-mers that appeared from mutated positions to disappear due to another mutation, and also prohibit the duplication of k-mers in independent branches of the phylogenetic tree. We will formalize the extended assumptions with a modification of an ISM tree that we call a *k-mer-based ISM tree*.

Definition 13 (k-mer-based ISM tree)**.** Let $C = \{G_1, \ldots, G_n\}$ be a genome collection composed of a set of equal length genomes. We call a rooted weighted binary tree $\mathcal{T}$ a *k-mer-based ISM tree* of the collection if

1. the root of $\mathcal{T}$ contains no mutated positions,
2. the leaves of $\mathcal{T}$ are exactly $\{G_1, \ldots, G_n\}$,
3. every edge is labeled by a set of mutations, and any two mutations in $\mathcal{T}$ are at least $2k$ positions apart,

4. each descendant inherits the mutations accumulated on all edges from the root to that node,
5. in each genome, any k-mer appears at most once,
6. with each mutation in a genome, k novel k-mers appear in that genome, where novel means such k-mer that only appears in the descendants of the given genome,
7. edge weights are defined as the number of mutations in the corresponding edge label.

Under these additional assumptions, we show that k-mer matrices of genome collections are ISM-compliant.

Theorem 3. Let $C = \{G_1, \ldots, G_n\}$ be a genome collection of genomes forming the leaves of a k-mer-based ISM tree. Then the k-mer matrix K built on the collection C is ISM-compliant.

Proof. The proof will be very similar to the case of SNP matrix. Let us call the corresponding k-mer matrix $K \in \{0,1\}^{m \times n}$, where m is the number of distinct k-mers observed in the collection. For each row i in K, define the subsets of column indices $T_i = \{j \mid K_{ij} = 1\}$, $F_i = \{j \mid K_{ij} = 0\}$. We want to show that for each pair of rows r, s, at least one of the four intersections $T_r \cap T_s, T_r \cap F_s, F_r \cap T_s, F_r \cap F_s$ is empty.

Any row that is all-ones or all-zeros automatically satisfies this condition, so we restrict attention to the remaining (nontrivial) rows. Under the k-mer-based ISM tree, each k-mer is incident with exactly one edge of the tree. This edge partitions the leaf set into two complementary subsets: those genomes that contain the k-mer (corresponding to T_i) and those that do not (corresponding to F_i).

Let us choose two nontrivial rows r, s in K. The edge associated with r divides the tree into subtrees with leaf sets corresponding to T_r and F_r. The edge associated with s must lie either within the subtree whose leaves are T_r or within the subtree whose leaves are F_r. Thus, there are four possible scenarios.

- If the edge of s lies within T_r, then either
  - $F_s \subseteq T_r$, hence $F_s \cap F_r = \emptyset$, or
  - $T_s \subseteq T_r$, hence $T_s \cap F_r = \emptyset$.
- If the edge of s lies within F_r, then either
  - $F_s \subseteq F_r$, hence $F_s \cap T_r = \emptyset$, or
  - $T_s \subseteq F_r$, hence $T_s \cap T_r = \emptyset$.

In all cases, at least one of the four intersections is empty. Therefore, the matrix K is ISM-compliant. □

Unitig matrix

We will further utilize the assumptions in Definition 13 to show that unitig matrices built on the data from k-mer-based ISM tree are ISM-compliant as well. By Definition 2, unitigs are created by compacting k-mers on non-branching path sharing the same color set in the corresponding de Bruijn graph. Therefore, the unitig matrix can be obtained from the k-mer matrix by removing some duplicate rows. It is easy to see that ISM-compliance is invariant to such transformation.

Lemma 3. Let M be an $m \times n$ ISM-compliant matrix, and let M_I be the submatrix of M formed by keeping only the rows indexed by $I \subseteq \{1, \ldots, m\}$. Then M_I is also ISM-compliant.

Proof. Since M is ISM-compliant, for every pair of rows r, s in M at least one of the intersections $T_r \cap T_s, T_r \cap F_s, F_r \cap T_s, F_r \cap F_s$ is empty. Removing rows has no impact on the pairwise comparison of the remaining ones, therefore the condition also holds for any subset of rows, including those forming the submatrix M_I. Hence M_I is ISM-compliant. □

Corollary 1. Let $C = \{G_1, \ldots, G_n\}$ be a genome collection of genomes forming the leaves of a k-mer-based ISM tree. Then the unitig matrix U built on the collection C is ISM-compliant.

Bloom-filter matrix

Next, we studied the ISM-compliance of a Bloom-filter matrix (Definition 7). In particular, ISM-compliance is surely guaranteed in the idealized case when a single hash function is used and collisions are absent, in which case the Bloom-filter matrix reduces to a k-mer matrix (up to possible empty rows). However, when multiple hash functions are employed, collisions of the form $h_m(k_i) = h_n(k_j)$ may occur, potentially violating ISM-compliance. It would be possible to show the ISM-compliance of a Bloom-filter matrix with several hash functions requiring that their outputs remain collision-free across all k-mers and functions, however, this would in practice be equivalent to assuming a single injective hash function as we do here.

Theorem 4. Let $C = \{G_1, \ldots, G_n\}$ be a genome collection forming the leaves of a k-mer-based ISM tree. Let B be a Bloom-filter matrix built on the collection C with $\mu = 1$ and m such that the probability of collisions of k-mers in B is negligible. Then B is ISM-compliant.

Proof. We will show that under these assumptions, the Bloom-filter matrix is very similar to the k-mer matrix. As the outputs of the hash function never collide for distinct k-mers, each row of the matrix is associated with at most one k-mer, and the binary values encode its presence and absence across genomes. The ISM-compliance is therefore implied by Theorem 3. □

Lastly, we note that collisions within a single hash function generally break the ISM-compliance. If two distinct k-mers are mapped to the same position, their encodings are combined into a single row of the Bloom-filter matrix. Given that the colliding k-mers might be from any two parts of the k-mer-based ISM tree, it is possible to observe two rows in the matrix that would break the ISM-compliance condition.

3.3 RLE binary matrix compression problem (RBMC)

We specified binary matrices as the data structure in our model of phylogenetic compression, now we want to model their compression. We choose the run-length encoding (RLE) as our low-level compression method. We formalize the general problem of RLE for binary matrices representing genome collections as follows. Consider a collection of genomes represented by a binary matrix, where each column corresponds to an individual genome. We will consider that RLE compression is performed row-wise: within each row, we identify consecutive runs of equal characters, and each run is represented by its length. For simplicity, each row is processed independently, and thus every row begins with a new run. Without loss of generality in our case, we assume that run length will always fit within a standard integer type. We will measure the final compressed size of the matrix as the total number of runs across all rows.

In the specific case of applying RLE to binary data, storing the symbol of each run is unnecessary once the first symbol in a row is known. In practice, the compressed representation of a binary matrix consists of two components: the first column, which determines the starting symbol of every row (and can itself be RLE-compressed), and the sequence of run lengths for each row, which fully specifies the remaining entries. In our theoretical model, we focus solely on the number of runs across all rows as the measure of compressed size for simplicity. If we wished to reconstruct the full matrix from the RLE representation, the first column would need to be stored in addition to the run lengths.

Example 8. Consider the following binary matrix A

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

The first row 101 contains three runs of length 1. The second row 000 contains one run of length 3. Similarly, the last two rows contain runs of lengths 2, 1 and 1, 1, 1. The resulting size of the RLE-compressed matrix A is the total number of runs, i.e., $3 + 1 + 2 + 3 = 9$.

With this encoding, we define the computational problem of minimizing the compressed size by reordering the input dataset in Problem 2. Specifically, we ask: what is the complexity of permuting the matrix columns to minimize its run-length encoding (RLE) compressed size? The objective is to find a column permutation that yields the smallest total number of row-wise runs. We call this optimization task the RLE binary matrix compression (RBMC) problem.

Problem 2 (RBMC).
Input: Binary matrix $A \in \{0,1\}^{m\times n}$
Output: Column permutation of A that minimizes the total number of row-wise runs

3.3.1 NP-hardness of RBMC

When we have no additional assumptions about the input matrix, the RBMC problem is NP-hard. It reduces to the path variant (without returning to the starting point) of the Traveling Salesperson Problem (TSP).

Theorem 5. The RBMC problem is NP-hard.

Proof. Given a binary matrix $A \in \{0,1\}^{m\times n}$, we construct a complete graph $G = (V, E)$ where each vertex $v_i \in V$ is the i-th column of A. For every pair $i \neq j$, the edge $(v_i, v_j) \in E$ is assigned a weight equal to the Hamming distance $w(v_i, v_j) = \mathrm{Ham}(v_i, v_j)$.

The row-wise run count in A can be directly expressed as $\sum_{j=1}^{n-1} w(v_j, v_{j+1}) + m$, where m is the height of A, which is an additive constant with respect to the column order. Therefore, finding the order of columns that minimizes the total run count is equivalent to finding the minimum-weight open path in G that goes through all nodes (i.e., Hamiltonian path), which is an instance of the path Traveling Salesperson Problem. To show that this problem is NP-hard under Hamming-distance weights, we reduce from the corresponding cycle version, which was proven to be NP-complete as the Hamming Traveling Salesperson Problem (HTSP) in [31].

We want to show that we can use the solution of the path TSP to solve the cycle variant. If we were able to compute the path solution in polynomial time, it would contradict the NP-hardness of the cycle variant.

Consider an instance of this cycle TSP variant given by our set of binary vectors $\{v_1, \ldots, v_n\}$ forming the columns of the matrix A and the graph G weighted by their pairwise Hamming distances. As described above, the shortest Hamiltonian path in G is exactly an ordering of the vectors that minimizes the total number of row-wise runs in A.

To connect the shortest-cycle and shortest-path variants, we use a distance-equalization argument. We fix one of the vectors, without loss of generality v_1, and modify its distances so that it has an equal distance to any other vector, while preserving the order of vertices in the shortest Hamiltonian path. Let $D = \max_{i\neq 1} \mathrm{Ham}(v_1, v_i)$ be the maximum Hamming distance between v_1 and any other vector. For each $v_i, i \neq 1$, we add a constant $L - \mathrm{Ham}(v_1, v_i)$ to each of its distances $\mathrm{Ham}(v_i, v_j), j \neq i \wedge j \neq 1$. For binary vectors, we can do so by appending the same fixed binary suffix of appropriate length to every vector. Because adding the same constant to all edges incident on a vertex uniformly increases every tour length by twice that constant, the city order of the optimal tour over all cities remains unchanged.

Once v_1 is equidistant from every other vertex, it can be removed without affecting the relative ordering of any optimal path on the remaining $n-1$ vertices. Solving this modified instance as a path TSP yields a shortest path on the vertices $v_2, \ldots, v_n$. Reinserting v_1 between the endpoints of this path closes it into a valid HTSP cycle on all n vertices. Thus, any HTSP instance reduces in polynomial time to the path version, proving this version NP-hard as well. Consequently, the RBMC problem is NP-hard. □

3.3.2 Computation of best and worst orders

The reduction to the TSP provides a direct approach for an RBMC solution via TSP solvers: compute the pairwise column distances and find the optimal ordering using a TSP solver such as Concorde [56]. Moreover, we can use to same approach to obtain also the worst possible ordering that yields the maximum total number of row-wise runs.

Lemma 4. Let A be a binary matrix and let G be the complete graph on its columns with edge weights $w(v_i, v_j) = \mathrm{Ham}(v_i, v_j)$. Let $W_{\max} = \max_{i\neq j} w(v_i, v_j)$ and define modified weights $w_m(v_i, v_j) = W_{\max} - w(v_i, v_j)$. Then the worst-case RBMC solution is obtained by solving the shortest-path problem under the modified weights w_m.

Proof. Under the modified weights, for any open Hamiltonian path $P = (v_1, v_2, \ldots, v_n)$, its total modified weight is

$$\sum_{k=1}^{n-1} w_m(v_k, v_{k+1}) = \sum_{k=1}^{n-1} \big(W_{\max} - w(v_k, v_{k+1})\big) = (n-1)W_{\max} - \sum_{k=1}^{n-1} w(v_k, v_{k+1})\,.$$

Since $(n-1)W_{\max}$ is a constant, minimizing this modified weight is equivalent to maximizing the sum of the original Hamming distances along the path. By the same argument as in the proof of Theorem 5, the sum of Hamming distances is the total number of runs in A up to an additive constant. Hence, a shortest path under w_m gives a column ordering of A with maximum total run count. □

Although solving the RBMC optimization problem is NP-hard for general binary matrices, we focus specifically on ISM-compliant matrices. In the next chapter, we show how the underlying tree structure of such data can be leveraged to solve the RBMC problem efficiently.

3.4 Neighbor Joining (NJ) solves the RBMC problem optimally for ISM-compliant matrices

When the binary matrix is ISM-compliant, the RBMC problem simplifies dramatically. The key property is the additivity of the Hamming distances proven in Theorem 1, which ensures that there is a tree graph explaining the distances. This breaks the NP-hardness and allows for a simpler solution than using a TSP solver. The central question is whether the tree describing the distances can be reliably recovered.

We will use the distance-based method called Neighbor Joining (NJ) (Section 2.3.2). The key property of NJ is that when the provided distances are additive, it is guaranteed that the NJ algorithm reconstructs the correct unique unrooted weighted tree topology that describes the distances. Moreover, the NJ algorithm is robust: even for non-additive distances, it reconstructs a tree minimizing the balanced minimum evolution criterion [57].

Lemma 5 ([57])**.** Let T_u be an unrooted weighted tree with shortest branch length s, and let D_{T_u} be the set of pairwise leaf distances of T_u. Let D_m be a set of distances from D_{T_u}, modified by some amount. If all pairwise distances in D_m deviate from their true values in D_{T_u} by less than $s/2$, then the NJ algorithm reconstructs T_u exactly. In particular, if the distances are additive, exact reconstruction is guaranteed.

This property provides a powerful tool for solving the RBMC problem on ISM-compliant matrices: not only are the Hamming distances among the matrix columns additive, but we are also able to obtain the tree topology itself in $\mathcal{O}(n^3)$ time. Once the tree is known, finding an optimal column permutation becomes equivalent to identifying the shortest open Hamiltonian path in that tree. Such a path can be obtained by a simple depth-first traversal, with the only remaining degree of freedom being the choice of the two endpoint leaves that minimize the total path length. We formalize this in the following theorem.

Theorem 6. Let M be an ISM-compliant matrix. The left-to-right order of leaves in the tree T inferred by Neighbor Joining from the pairwise Hamming distances of the columns of M is an optimal solution of the RBMC problem on M up to an additive term bounded by the diameter of T (i.e., the longest leaf-to-leaf path in T).

Proof. By Theorem 1, the Hamming distances between the columns of M are additive, therefore Lemma 5 guarantees that the NJ algorithm produces a weighted unrooted tree T whose leaf distances match the Hamming distances in M. By the proof of Theorem 5, finding a column permutation of M that solves the Problem 2 optimally is equivalent to finding a minimum-weight Hamiltonian path through the leaves of T. Since T is the minimal spanning tree of all leaves, its shortest leaf treaversal is realized by a standard depth-first traversal (DFT).

A DFT that starts at leaf u and ends at leaf v visits each edge twice, except those on the unique $u \to v$ path, which are traversed only once. If

$$S = \sum_{e \in E(T)} w(e)$$

is the total weight of T, then the length of the DFT route is

$$L(u, v) = 2S - d(u, v)$$

where $d(u, v)$ is the distance between u and v in T.

The DFT length can be further optimized by maximizing $d(u, v)$, i.e., by choosing the two leaves that are farthest apart as endpoints of the path. The best possible traversal length is therefore

$$L(u, v) = 2S - D,$$

where $D = \max_{\{u,v\}} d(u, v)$ is called the diameter of T. □

This reflects the intuition provided by phylogenetic compression. The methods described here represent a simplified protocol of phylogenetic compression (Figure 3.1d): first, we represent our data modeled by the ISM as an ISM-compliant matrix (e.g., k-mer matrix), then we infer the phylogeny of the collection from the binary vectors using NJ, and lastly, we reorder the columns of the matrix according to the phylogenetic tree left-to-right before applying the RLE compression. In this protocol, Theorem 6 explains that the compression guided by phylogeny gives optimal results.

Note that if we were instead interested in the shortest *closed* Hamiltonian path on the leaves (i.e., a traversal that returns to its starting point), any left-to-right enumeration of the leaves would be optimal, regardless of how the tree branches are rotated. In this case, the depth-first traversal always has total length $L(u, v) = 2S$, since every edge is traversed exactly twice. In the terminology of the row-wise RLE-compression matrix problem, this corresponds to applying RLE to a "cylindrical" matrix, where first and last character of each row are considered adjacent.

Lastly, an important property holds for the trees inferred by NJ from ISM-compliant matrices. We have shown that SNP, k-mer, unitig, and Bloom-filter matrices are all ISM-compliant, and therefore NJ applied to any of them recovers a tree whose left-to-right leaf order yields optimal RLE compression. However, this does not yet clarify how the trees inferred from these different matrix representations relate to one another. The following theorem shows that, under ISM-compliance, all of these matrices induce exactly the same tree topology, differing only in edge weights. This fact is crucial for our experimental pipeline: once a phylogenetic tree is inferred, the resulting genome order can be reused for all others.

Theorem 7. Let $C = \{G_1, \ldots, G_n\}$ be a collection genomes forming the leaves of a k-mer-based ISM tree. Let S, K, U, and B denote the SNP, k-mer, unitig, and Bloom-filter matrices constructed from C, where the SNP matrix uses a compatible reference genome, and the Bloom-filter matrix is built with $\mu = 1$ and m chosen so that the false-positive rate is negligible. Then the tree constructed by the Neighbor-Joining algorithm from the Hamming distances of S, K, U, or B has the same tree topology (up to branch rotations), with leaves corresponding to the genomes in C.

Proof. Under the ISM, each mutation occurs exactly once on the underlying evolutionary tree. Consider any edge e of this tree, and let $M(e)$ denote the number of mutations that occur along e. For any two genomes G_i and G_j, their Hamming distance in each of the four matrix types is the sum, over all edges on the path between G_i and G_j, of a positive contribution per mutation on that edge. The magnitude of this contribution depends on the matrix type.

For the SNP matrix S, each mutation contributes exactly 1 to the Hamming distance between any pair of genomes whose path contains the corresponding edge. Thus the induced distances are additive with edge weights proportional to $M(e)$.

For the k-mer matrix K, by Definition 13, each point mutation creates exactly k new k-mers and deletes k old ones, contributing $2k$ to the Hamming distance for every pair of genomes separated by that mutation. Thus the induced distances are again additive, with edge weights equal to $2k\,M(e)$.

The unitig matrix U is obtained from the k-mer matrix by merging certain rows that are identical across all genomes. As shown in Lemma 3, merging duplicated rows preserves additivity of the induced distances. Concretely, in the k-mer-based ISM tree, each edge e corresponds to $2k\,M(e)$ rows that differ exactly on the genomes separated by e. When some of these rows are merged, the Hamming distances between genomes separated by e decrease by the number of merged rows, which corresponds to reducing the weight of edge e by the same amount. Since only duplicated rows are merged, the resulting edge weights remain strictly positive. Thus the unitig distances correspond to the same additive tree topology as the k-mer distances.

The Bloom-filter matrix B with $\mu = 1$ and negligible false-positive rate is very similar to the k-mer matrix as explained in the proof of Theorem 4 up to empty rows, which do not contribute to the Hamming distances. The lengths of the tree branches are therefore also equal to $2k\,M(e)$.

In all four cases, the pairwise Hamming distances are additive with respect to the same underlying tree topology, differing only by a scaling factor or by reductions in branch lengths that preserve positivity. It follows that NJ applied to S, K, U, or B reconstructs the same unrooted tree (up to branch rotations). □

Chapter 4

Implementation of experimental pipeline

The goal of this chapter is to describe the pipeline that we developed to evaluate how different column orderings influence the run-length compressibility of binary bacterial genome matrices. Our objective is to understand which ordering strategies produce the most compact RLE compression, and how these strategies behave across datasets, matrix types, and k-mer sizes. To answer these questions in a reproducible and scalable way, we implemented an evaluation pipeline using Snakemake [58]. The workflow coordinates all steps required to construct genome matrices, compute pairwise Hamming distances between columns, generate optimal and worst-case orderings via TSP formulations, infer phylogeny-based orderings, and finally quantify the resulting RLE-compressed sizes. The pipeline is modular and supports experiments across multiple datasets, k-mer sizes, and matrix types.

4.1 Overview of the evaluation pipeline

We conducted our experiments on several datasets of bacterial genomes obtained from public repositories of high-quality assemblies in FASTA format (Appendix B). For each dataset, we first compiled a list of available assemblies and then extracted a random subset of 1000 genomes that served as the input of the pipeline. No additional preprocessing of the FASTA files was performed, as all subsequent steps of the pipeline operate directly on the assemblies.

To compute the unitigs of the collections, we used Fulgor [14], a software tool for indexing large collections of genomes by constructing a colored de Bruijn graph and compacting it into unitigs. Each genome corresponds to one color, and the resulting index can be exported into *dump files* containing the unitig sequences and their associated color sets.

We chose three types of binary matrices to represent the genome collections: the k-mer matrix, the unitig matrix, and the unique-row matrix. The k-mer and unitig matrices follow Definition 5 and Definition 6, respectively. In the unique-row matrix, each row corresponds to one color set, i.e., to a set of genomes sharing exactly the same unitigs. Theoretically, it can be derived from the unitig matrix by collapsing all identical rows. We showed that all three matrices are ISM-compliant when constructed from ISM-compliant data (Theorem 3, Lemma 3, Corollary 1). This motivates their use as test cases for evaluating the robustness of ISM-based predictions under realistic bacterial evolution.

To obtain column orderings that minimize or maximize the run-length-encoded size of the binary matrices, we applied the reduction of Problem 2 to the TSP to be able to use an optimalized TSP solver Concorde [56]. We used it also to compute the worst possible solution using distance modification from Lemma 4.

For phylogenetically guided column orderings, we inferred approximate phylogenetic trees for each dataset and extracted their left-to-right leaf orders. We employed two classical distance-based reconstruction methods implemented in Attotree [59]: Neighbor Joining (NJ), which is guaranteed to recover the correct topology for ISM-compliant matrices in our theoretical model (Theorem 6), and the Unweighted pair group method with arithmetic mean (UPGMA), which requires a stronger assumption of ultrametric distances (Definition 9) to return the true tree. These two methods provided the phylogenetic heuristics for ordering matrix columns in

our experiments.

In addition to phylogeny-based and TSP-based orderings, we included a randomized ordering, which served as a baseline model against which to compare the structured orderings.

To quantify how different column orderings affect the compressibility of the binary matrices, we measured the total number of row-wise runs as defined in Section 3.3. For a binary matrix $M \in \{0,1\}^{m\times n}$ and a fixed column ordering $\pi = (\pi_1, \ldots, \pi_n)$, the total number of runs is

$$\sum_{j=1}^{n-1} \mathrm{Ham}(M_{\bullet,\pi_j}, M_{\bullet,\pi_{j+1}}) + m,$$

where m is the number of rows. The first term counts all bit changes across adjacent columns, and the additive constant accounts for the initial run in each row.

We also compared this row-wise measure with an alternative RLE strategy that linearizes the entire matrix and applies RLE globally. In this case, the total number of runs becomes

$$\sum_{j=1}^{n-1} \mathrm{Ham}(M_{\bullet,\pi_j}, M_{\bullet,\pi_{j+1}}) \;+\; \sum_{i=1}^{m-1} \mathrm{Ham}(M_{i,\pi_n}, M_{i+1,\pi_1}) \;+\; 1,$$

which removes the additive constant but introduces an additional term capturing transitions between the last column of one row and the first column of the next. Although we computed both quantities for all experiments, the differences between them were nearly constant and did not affect the qualitative conclusions. We therefore comment only the row-wise run counts results in Chapter 5, which align with our theoretical model. A comparison of the two RLE strategies is shown in Appendix C.

4.2 Components of the evaluation pipeline

4.2.1 Snakemake workflow

Snakemake is a workflow management system that provides a declarative way to specify computational steps, their dependencies, and the files they produce. It ensures that each rule is executed only when its inputs are available and up to date, and it automatically manages parallel execution and organization of intermediate results. In our implementation, Snakemake links together all components of the evaluation pipeline (Figure 4.1).

To support experiments across many parameter combinations, the workflow extensively uses *wildcards*. The primary wildcards include the dataset identifier (`dataset`), the k-mer size (`k`), the number of selected genomes (`N`), the matrix type (`type`), the phylogenetic inference method (`method`), and the TSP variant (`variant`). These wildcards allow Snakemake to automatically expand the workflow into all required jobs.

Reproducibility is supported through explicit storage of selected genome subsets, deterministic rule outputs, and the use of a dedicated `conda` environment for all software dependencies. The workflow was executed with `snakemake-minimal` 7.32.4, typically invoked with

```
snakemake -j 32 -resources concurrency=1 -latency-wait 30 -rerun-incomplete -keep-going
-show-failed-logs -reason -use-conda .
```

The pipeline relied on Python 3.10.19, Fulgor 4.0.0, Attotree 0.1.6, and the Concorde TSP solver distributed as a precompiled Linux binary (“Concorde for Red Hat Linux 8.0”).

4.2.2 Input and randomized nested subsets

As input, the pipeline uses plain-text `dataset.txt` files containing absolute paths to the genome FASTA files. To evaluate the impact of dataset size, nested subsets are generated from these lists in the `randomization` and `create_selection` rules. To avoid unintended ordering bias, each subset of size N is taken from a version of the original list that has been randomized using `sort -R`, after which the first N entries are selected via

```
head -n N {dataset}_randomized.txt | sort > {dataset}_N{N}.txt
```

and stored as individual `{dataset}_N{N}.txt` files for reproducibility. Further in the pipeline (Section 4.2.4), for k-mer and unitig matrices, these subsets serve as a guide of which genome columns to pick from the global

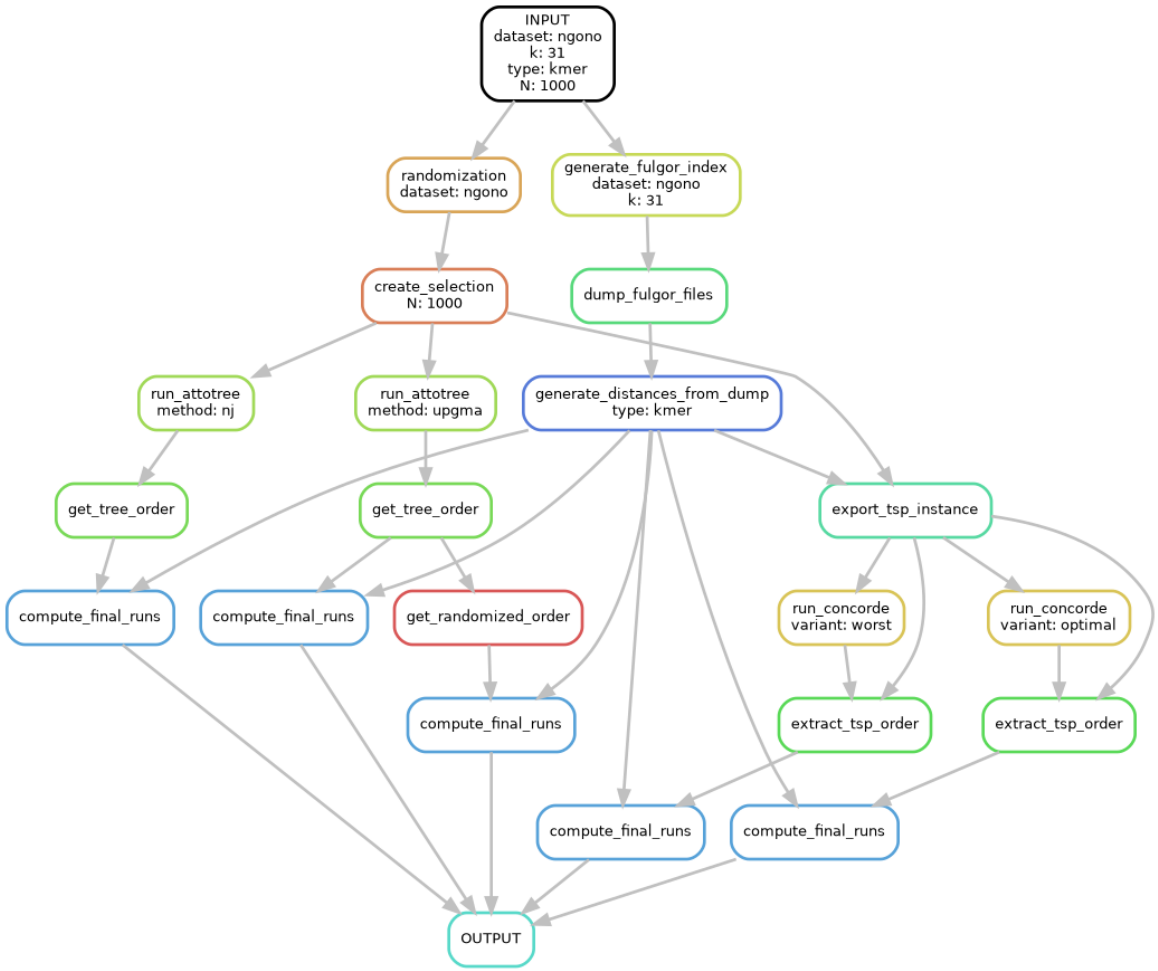


Figure 4.1: Example directed acyclic graph of the evaluation pipeline. This diagram shows the Snakemake workflow for a single dataset (`ngono`), a single k-mer size ($k = 31$), one matrix type (`k-mer`), and one dataset size ($N = 1000$). The graph illustrates the rules for genome selection, Fulgor index construction, unitig and color-set extraction, Hamming distance computation, TSP instance generation and solving, phylogenetic tree inference via Attotree, and final RLE run evaluation across multiple ordering strategies. Although this example omits other datasets and parameter combinations for simplicity, the implemented workflow generalizes to all configurations used in the experiments.

matrix to order, i.e., the distances are calculated on the matrix built from the whole dataset, and only the corresponding column distances are then used. In the case of unique-row matrices, these subsets are used as input at the beginning of the pipeline and the matrices are rebuilt again for each of the subsets to ensure that rows are truly unique.

4.2.3 Unitig sets using Fulgor

Unitigs are computed using Fulgor. Given a k and an input `{dataset}.txt` file (or `{dataset}_{N}.txt` file for unique-row matrices) listing the genome assemblies, the index is built in the rule `generate_fulgor_index` using

```
fulgor build -l {dataset}.txt -o 04_indices/{dataset}_k{k} -k {k} -m {m}
-d 04_indices/{dataset}_k{k}_tmp -g 32 -t 32 -verbose -check
```

where `m` denotes the minimizer size, chosen proportionally to k following Fulgor's recommended settings. The corresponding dump files are produced in the rule `dump_fulgor_files` via

```
fulgor dump -i {dataset}_k{k}.fur -o 05_dumps/{dataset}_k{k}
```

which generates three files containing the information about the unitigs presence or absence in the genomes:

- `metadata.txt` provides global metadata, including the value of k, the number of colors (equal to the

number of genomes), the total number of unitigs, and the number of distinct color sets. A color set is the set of genomes in which a given unitig appears.

- `color_sets.txt` lists all color sets, one per line, in the format

  `color_set_id=0 size=5 62 174 617 648 898`.

  The trailing integers correspond to the indices of genomes in the input `.txt` file.

- `unitigs.fa` contains FASTA-formatted unitig sequences, each annotated with its unitig identifier and the identifier of its associated color set, e.g.,

  ```
  > unitig_id=0 color_set_id=0
  CAGACCGCCTTCGACGATCCAGTTCTGGCCTTGCATGCCGGT
  ```

  This file links each unitig to the genomes in which it occurs.

4.2.4 k-mer, unitig and unique-row matrix distances

Instead of constructing these matrices explicitly, the pipeline computes all pairwise Hamming distances between their columns directly from the dump files using a Python script in the rule `generate_distances_from_dump`. The structure of the matrices is fully determined by the dump files:

- The unitig matrix has one row per unitig in `unitigs.fa` and one column per genome. A matrix entry is 1 if the genome index appears in the corresponding color set in `color_sets.txt`.
- The k-mer matrix is defined analogously, but with one row per distinct k-mer. Because the Fulgor index is constructed for a fixed value of k, the number of k-mers contained in a unitig is determined by the unitig's length, as each unitig is formed from overlapping k-mers. In a colored de Bruijn graph, these k-mers are unique within each unitig, so the k-mer matrix can be obtained by duplicating the row of a given unitig x times, where x is the number of k-mers in that unitig.
- The unique-row matrix is the simplest to build, as each unique row corresponds exactly to one color set in `color_sets.txt`, which determine the 1-entries of that row.

The `unitigs.fa` file is processed in a streaming fashion and for each constructed row, the distance of genomes whose entries differ is increased in a global distance matrix. This yields the full distance matrix for n genomes.

To handle large datasets efficiently, the computation is parallelized across available threads and uses bit-parallel operations to process blocks of rows at a time. The overall computational complexity is $\mathcal{O}(mn^2)$, where m is the number of rows (unitigs, k-mers, or unique rows) and n is the number of genomes. The resulting distance matrices are stored as text files and compressed using `xz -9` to reduce disk usage.

4.2.5 TSP instance generation and solving

TSP instances are generated by a Python script in the rule `export_tsp_instance`, which takes as input the distance matrix and the list of selected genomes, extracts the relevant submatrix, and creates two TSP instance files

- an *optimal* variant, where edge weights are equal to the Hamming distances D_{ij}, yielding a minimization problem;
- a *worst-case* variant, where edge weights are transformed to $D_{max} - D_{ij}$, converting the maximization objective into a minimization problem.

Because Concorde solves the cycle version of the TSP, each instance contains so-called *dummy node* with zero-weight edges to all other nodes. This ensures that the optimal cycle corresponds to an optimal path in the original graph, and the dummy node is removed after solving.

Both instances are written in TSPLIB format using the conventions

```
EDGE_WEIGHT_TYPE : EXPLICIT
EDGE_WEIGHT_FORMAT : FULL_MATRIX .
```

Here is an example of a generated instance file for 39 genomes:

```
NAME : 39 points
TYPE : TSP
COMMENT : keys: DUMMY_CITY_SEPARATOR,SAMEA3726531,SAMEA4362530,...
EDGE_WEIGHT_TYPE : EXPLICIT
EDGE_WEIGHT_FORMAT : FULL_MATRIX
NODE_COORD_TYPE : NO_COORDS
DISPLAY_DATA_TYPE : NO_DISPLAY
EDGE_WEIGHT_SECTION
0 0 0 0...
0 0 1000000 5827...
...
```

The `COMMENT` field stores the mapping between node indices and genome identifiers, enabling reconstruction of the final ordering. In the `EDGE_WEIGHT_SECTION`, there is the full distance matrix between the given matrix columns. Concorde is invoked by the rule `run_concorde` with default parameters

```
concorde -o instance.sol instance.tsp
```

and the output file `instance.sol` contains the order of nodes producing the shortest cycle on the given instance. The corresponding column ordering is extracted by a Python script in the rule `extract_tsp_order`, which removes the dummy node and linearizes the cycle into a path. The final ordering is written to

```
10_orders/{dataset}_{variant}_k{k}_N{N}_{type}.txt ,
```

where `variant` can be optimal or worst, and `type` denotes the type of the matrix.

During the evaluation process, we encountered two practical limitations of Concorde that required additional handling. The first issue arose for subsamples smaller than $\tilde{3}0$ nodes. For such instances, Concorde does not invoke its full branch-and-cut computing but instead switches to a specialized Held–Karp dynamic-programming routine implemented in its function `CCheldkarp_small`, and these settings cannot be user-overriden. This routine is designed for speed but imposes strict internal limits the magnitude of individual edge weights. For some of our small subsamples, these limits were exceeded, causing Concorde to terminate with the error `edge too long`. To avoid triggering the small-instance code path, we artificially increase the instance size by duplicating the dummy node until the total number of nodes exceeded 35. Because all dummy nodes have zero distances between each other and identical distances to all non-dummy nodes, they form a contiguous block in any optimal tour and can be removed after solving without affecting the optimal ordering of the original nodes.

The second limitation concerns the magnitude of edge weights. When the Hamming distances between columns reach values on the order of millions, the resulting optimal tour length is of the order of billions. These values exceed the internal integer range used by Concorde, leading to arithmetic overflow and the error `OVERFLOW in CCbigguy_addmult`. To address this, the instance generating script applies a uniform scaling whenever the maximum distance D_{max} exceeds 10 000. Specifically, a scale factor $s = D_{max}/10\,000$ replaces each distance d_{ij} with $d_m = \max\{1, \mathrm{round}(d_{ij}/s)\}$, ensuring that all distances remain positive integers. This transformation may collapse distinct distances to the same integer value, potentially introducing minor perturbations in the minimization function. However, given the large range of the original distances and the qualitative nature of our comparison between ordering strategies, this approximation does not significantly affect the results of the pipeline.

4.2.6 Phylogenetic tree inference

Phylogenetic trees are constructed using Attotree [59], which integrates Mash sketching [22] for fast estimation of pairwise genomic distances, and QuickTree [60] for tree reconstruction. Given a list of genome assemblies, Attotree computes pairwise Mash distances and then applies either NJ or UPGMA to obtain an approximate phylogeny in the rule `run_attotree` via

```
attotree -t 32 -m {nj|upgma} -L {dataset}_N{N}.txt -V > {output.nw} ,
```

where `-t` specifies the number of threads, `{dataset}_N{N}.txt` contains the list of genomes in the selected subset, and the `output.nw` is the resulting Newick-formatted tree. All trees are inferred once per selection file using Attotree's default Mash sketch size (corresponding to $k = 21$), and the resulting leaf orderings are reused for all k-mer sizes. We assume that a fixed sketch size provides sufficiently accurate estimate of distances for the

phylogenetic reconstruction across k and datasets, given that the theoretical model shows that the tree topology is the same regardless of the matrix type (Theorem 7).

The left-to-right order of leaves is obtained from a preorder traversal in the rule `get_tree_order` by a Python script. It loads the Newick tree using `ete3` library and outputs the leaves in the order in which they are encountered, without applying any additional processing steps. Preorder traversal visits each internal node before its children and processes children from left to right.

The final leaf orderings were written to files of the form

`10_orders/{dataset}_{nj|upgma}_k{k}_N{N}_{type}.txt` .

4.2.7 RLE compression evaluation

The evaluation is performed by a Python script that implements the RLE formulas above in the rule `compute_final_runs`. The script takes as input the distance matrix for the whole dataset and the ordering of the selected genomes, and additionally requires the number of matrix rows m. This value was obtained from the Fulgor metadata file: for the unitig matrix, m equals the number of unitigs, for the k-mer matrix, the number of k-mers was computed from the unitig count, and for the unique-row matrix, m equals the number of color sets.

Random orderings were generated by shuffling the genome subset order obtained from the UPGMA tree by

`cat {dataset}_upgma_k{k}_N{N}_{type}.txt | sort -R > {dataset}_randomized_k{k}_N{N}_{type}.txt` .

The choice of which ordering to randomize was arbitrary. An equivalent randomized baseline could have been obtained from the TSP ordering or from the original subset order.

For each dataset, matrix type, k-mer size, and ordering method, the final output run counts were written to `.txt` files

`11_runs/{dataset}_{method}_k{k}_N{N}_{type}.runs` .

Chapter 5

Experimental evaluation

Real bacterial genome collections do not fully adhere to the ISM assumptions underlying our theoretical model. In practice, genomes are finite, individual nucleotide positions may mutate repeatedly or revert, many bacterial species undergo recombination and horizontal gene transfer, and genomic variation includes insertions, deletions, and structural rearrangements. These processes violate the idealized ISM framework and may influence the structure and compressibility of the resulting binary matrices.

To assess how robust the ISM-based predictions remain under such non-ideal evolutionary conditions, we assembled several bacterial genome datasets of varying expected adherence to ISM assumptions (Appendix B). The `ngono` dataset, consisting of 1000 *Neisseria gonorrhoeae* genomes, represents a relatively homogeneous, single-species collection with limited recombination, and therefore most closely approximates ISM-like evolution. The signal of vertical descent should dominate over homologous recombination or horizontal gene transfer. The `ngono-spneumo` dataset combines two distinct species in equal proportion, introducing higher divergence from ISM assumptions and more violations of additivity. This dataset better resembles a large bacterial genome collection, i.e., containing multiple species, each represented by many genomes, than a single-species collection. Finally, the `diverse` dataset, drawn from many diverse species of the phylogenetically compressed 661k collection, contains hundreds of species spanning broad evolutionary distances and represents a severe violation of the ISM assumptions. In particular, the assumption of each site mutating at most once becomes entirely invalid as the diverse collection lacks a clear vertical descent tree and adheres more to a network-like evolution. Consequently, the distances in the corresponding matrices are hardly additive. These datasets allowed us to evaluate how compression performance degrades as ISM assumptions are progressively violated.

With the evaluation pipeline described in Chapter 4, we constructed the Hamming distances of the columns of the corresponding k-mer, unitig, and unique-row matrices for each dataset and quantified their run-length compressibility after applying RLE. For every dataset, we compared multiple column-ordering strategies: the optimal and worst-case solutions of the TSP formulation, two phylogeny-based orderings derived from NJ and UPGMA trees, and a randomized baseline. We further examined how dataset size and k-mer size impact the observed compression behavior.

Because the datasets differ substantially in their evolutionary properties, we expect these differences to be reflected in their achievable compression gains. Single-species collections with limited recombination should more closely follow ISM-like behavior and therefore benefit strongly from phylogenetically informed orderings. In contrast, highly diverse datasets containing many unrelated species are expected to deviate from ISM structure, reducing the advantage of phylogenetic or TSP-derived orderings. Finally, by comparing random orderings not only to phylogenetic ones but also to the theoretical worst-case arrangement, we can compare how far real datasets lie from both ideal and worst-case scenarios.

5.1 The impact of dataset diversity

First, we asked how close the NJ-based RLE compression would be from the optimal solution across the different matrix types and dataset diversities. In the case of `ngono`, we found (Figure 5.1, first column) that all phylogeny-based reorderings remained within 3 % of the optimal and achieved roughly a 5× improvement over a randomized baseline on all three matrices. The worst-case ordering yielded around 120% of the randomized baseline, leaving the randomized baseline far closer to the worst-case scenario than the optimal (and phylogeny-

informed) one. In the `ngono-spneumo` case, for all matrices, the results showed qualitatively similar trends as in single-species dataset but with larger relative differences between the orderings (Figure 5.1, second column). The optimal solution improved randomized order by 10 fold and both NJ and UPGMA performed within 1 % range from the optimal solution. The worst-case ordering reached over 160 % of the randomized baseline and resulted from an order alternating the species' genomes. Finally, we found that for the `diverse` dataset (Figure 5.1, third column) the overall differences between the orderings were significantly smaller compared to the other datasets. In particular, the optimal solution resulted in over 81 % of randomized order on k-mer matrix, over 57 % on unitig one, and over 44 % on the unique rows, thus although tree-guided orders still approached the optimum with less than 1 % difference, absolute gains were relatively low compared to less diverse datasets. This suggests that phylogenetic ordering is robust to violations of ISM assumptions, and that the overall possible improvement depends on the diversity of the dataset. An interesting observation is that the absolute numbers of the runs in the unique-row matrix are much lower than the ones in k-mer or unitig matrices. This is probably caused by the diversity of the dataset, containing many different and generally short unitigs that are shared by small amounts of genomes, which causes many duplicated rows.

5.2 Comparison of NJ with other inference methods

Although NJ is fast in practice, especially in combination with sketching, its worst-case time complexity is cubic in the number of genomes, which can become prohibitive for very large collections. We were interested in whether simpler and faster phylogenetic methods could serve as practical substitutes, even though they rely on stronger assumptions about the evolutionary process. A natural candidate is UPGMA (Section 2.3.2), which assumes ultrametric distances and runs in quadratic time.

We incorporated UPGMA into all our experiments and found that it closely matches the performance of NJ across all datasets, in many cases even slightly outperforming it (Figure 5.1). For the single-species and two-species collections, the difference between NJ and UPGMA was within 1% relative to the randomized baseline. In the `ngono-spneumo` dataset, UPGMA produced marginally better compression than NJ, and in the `diverse` dataset it consistently outperformed NJ across all three matrix types. One possible explanation is that UPGMA's simple averaging rule captures the local similarity structure that is most relevant for minimizing the number of runs, even if its underlying evolutionary assumptions are unrealistic. In other words, although UPGMA is not a biologically accurate model for our datasets, it may still recover the geometric relationships between genomes that matter in our experiments for RLE compression.

Furthermore, we also compared NJ with a maximum-likelihood (ML) phylogeny. Although we did not integrate ML inference directly into our pipeline, we obtained a dataset of $N = 1102$ *N. gonorrhoeae* genomes, referred to as `ngono-rase`, together with a precomputed phylogenetic tree. This tree was inferred using a RAxML maximum-likelihood pipeline (Section 2.3.2) applied to a recombination-filtered core genome alignment [25].

Using this RAxML tree as an external benchmark, we evaluated the k-mer matrix of the `ngono-rase` dataset for $k = 31$. All three phylogenetic orderings (NJ, UPGMA, and RAxML) lie within 2% of the optimal TSP solution, which compresses the k-mer matrix to 14% of the randomized baseline (Figure 5.2). Interestingly, the RAxML-derived ordering performs slightly worse than NJ, despite being biologically more accurate. This again supports the observation that RLE compression performance in these experiments depends primarily on capturing the local geometric structure of genome space, rather than on reconstructing a more accurate evolutionary history.

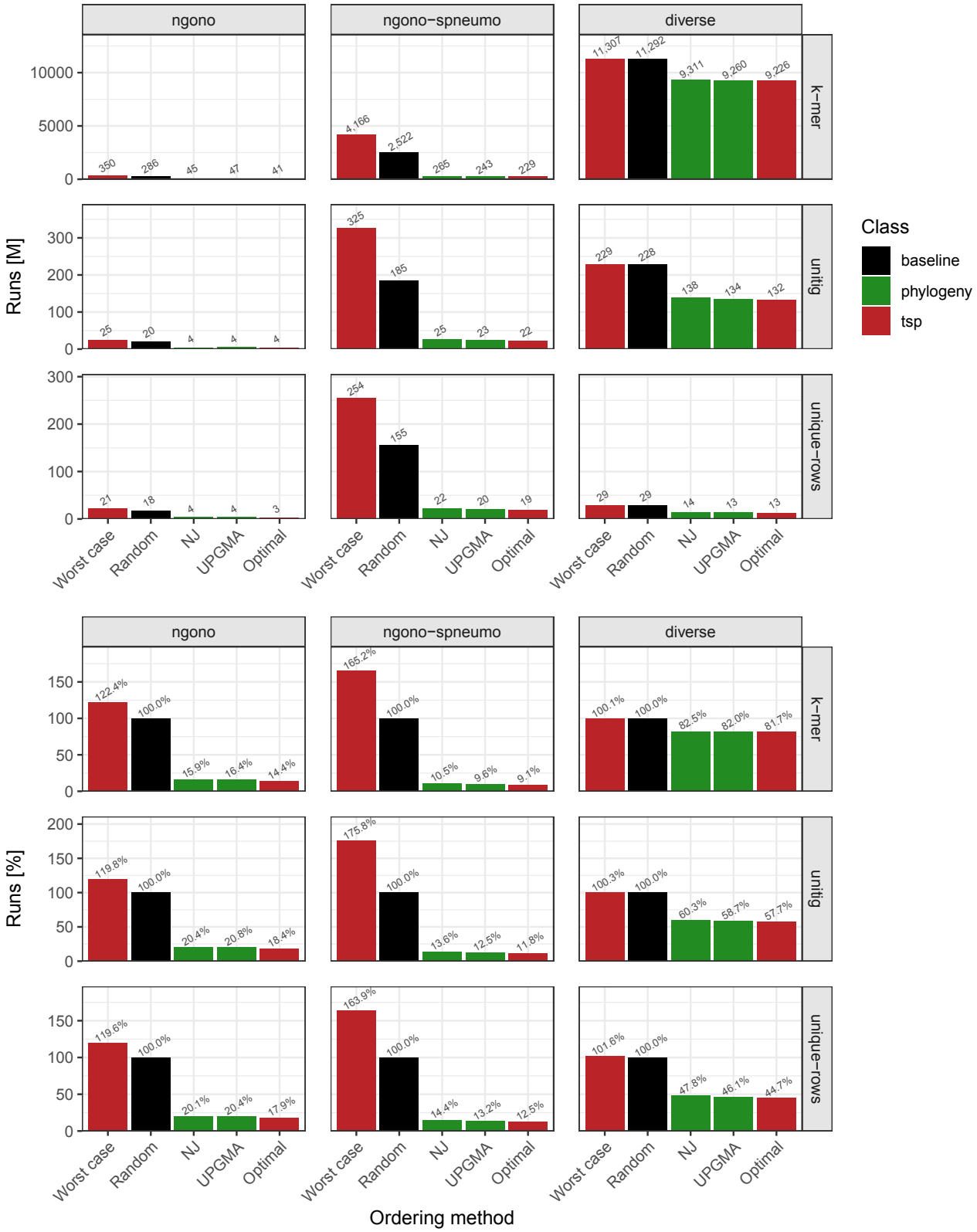


Figure 5.1: The impact of genome order on the size of RLE compression of k-mer, unitig and unique-row matrices for three datasets of varying diversity. The analysis was done for three datasets from Appendix B of size $N = 1000$ genomes and for $k = 31$, with respect to reordering of the genomes by random order (baseline), phylogenetic orders obtained from NJ and UPGMA, and the worst and best possible orderings computed exactly by a TSP solver.

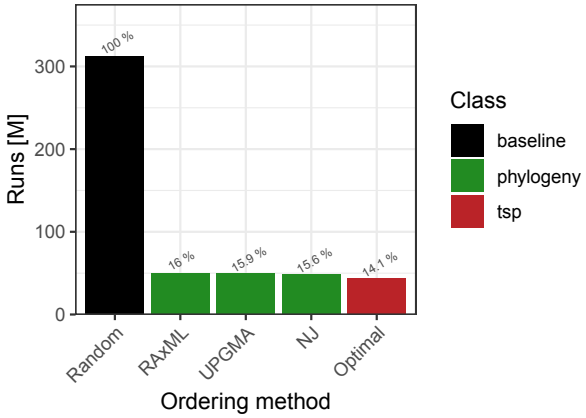


Figure 5.2: Comparison of NJ with other phylogenetic inference methods. We evaluated the RLE compression of the `ngono-rase` dataset using NJ and UPGMA within our pipeline and compared these results with the ordering obtained from an external maximum-likelihood tree inferred by RAxML.

5.3 Impact of the dataset size

We also examined the impact of the dataset size n on the improvement in compression with respect to the dataset diversity. We expected that larger single-species collections would yield greater improvements, while more diverse datasets would show more modest gains, with phylogeny-guided orderings remaining near-optimal regardless of size and diversity. As described in Chapter 4, for k-mer and unitig matrices the pipeline computed Hamming distances once in the matrix of the whole dataset, and nested subset runs were evaluated by extracting the corresponding columns to obtain the optimal, worst-case, and phylogeny-guided tours. The height of the k-mer and unitig matrix was therefore constant across all experiments. In the case of the unique-row matrix, the matrix was recomputed for each subsample before applying the rest of the pipeline.

Across all datasets and sample sizes (Figure 5.3), NJ-guided orderings remain consistently close to the optimal solution, typically within a few percentage points. Compression improves steadily as n increases, reflecting an initial saturation phase, where longer runs can form only once most k-mers or unitigs become present in the dataset subset. In single-species and two-species collections this saturation occurs quickly, whereas in the `diverse` dataset it appears only after roughly 100 genomes. Worst-case orderings are typically close to the random baseline, except in the two-species dataset, where the worst-case arrangement resulting from alternating genomes from the two species produces substantially poorer compression, with the gap widening as n increases.

The RLE compressibility of the k-mer, unitig, and unique-row matrices is generally similar. A notable exception occurs in the `diverse` dataset, where the unique-row matrix becomes more compressible earlier than the k-mer and unitig matrices. We conjecture that this difference is partly an artifact of the experimental design: because the unique-row matrix is rebuilt for each subsample, its structure reflects only the selected genomes, while the k-mer and unitig matrices include empty rows corresponding to k-mers or unitigs absent from the subset. On the other hand, these empty rows contribute only a constant baseline number of runs that is present across all subsets, and other types of runs that do not include this constant showed similar trends across matrix types (Figure C.4).

5.4 Impact of k-mer size

Lastly, we sought to evaluate the impact of k on compressibility, particularly in terms of how different k values affect the performance of phylogeny-guided reordering. We anticipated that shorter k-mers would lead to more severe violation of the ISM assumptions, in particular that each k-mer appears at most once per genome and that k-mers emerging due to mutations are novel and unique. Due to the low specificity of short k-mers, they are more likely to occur in multiple, unrelated genomic contexts. In the extreme case of very short k, the matrices become nearly saturated, since most k-mers are present in nearly all genomes. Therefore, we expected the compression results to dramatically improve for larger k's, where most k-mers become unique and lead to sparser matrices which are more likely to follow the assumptions of the ISM.

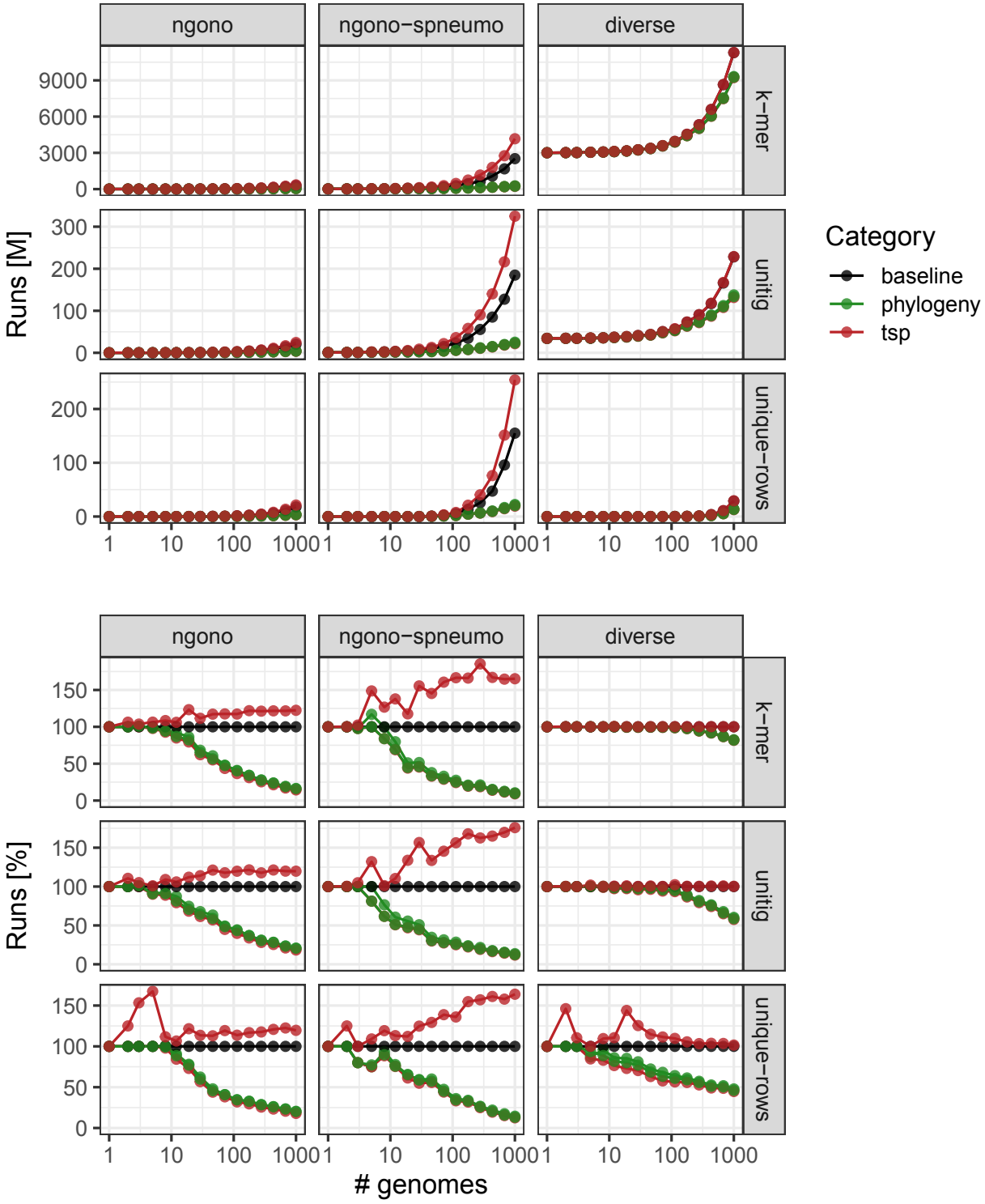


Figure 5.3: Impact of dataset size on compression performance. For each dataset, We evaluated the impact of increasing the number of genomes n on the row-wise run count in a RLE-compressed binary matrix. We generated nested subsamples of n genomes up to full dataset size and computed optimal, worst-case, random and phylogeny-guided orderings for each subset. The results show how compression improves with increasing sample size and how this behavior varies with dataset diversity.

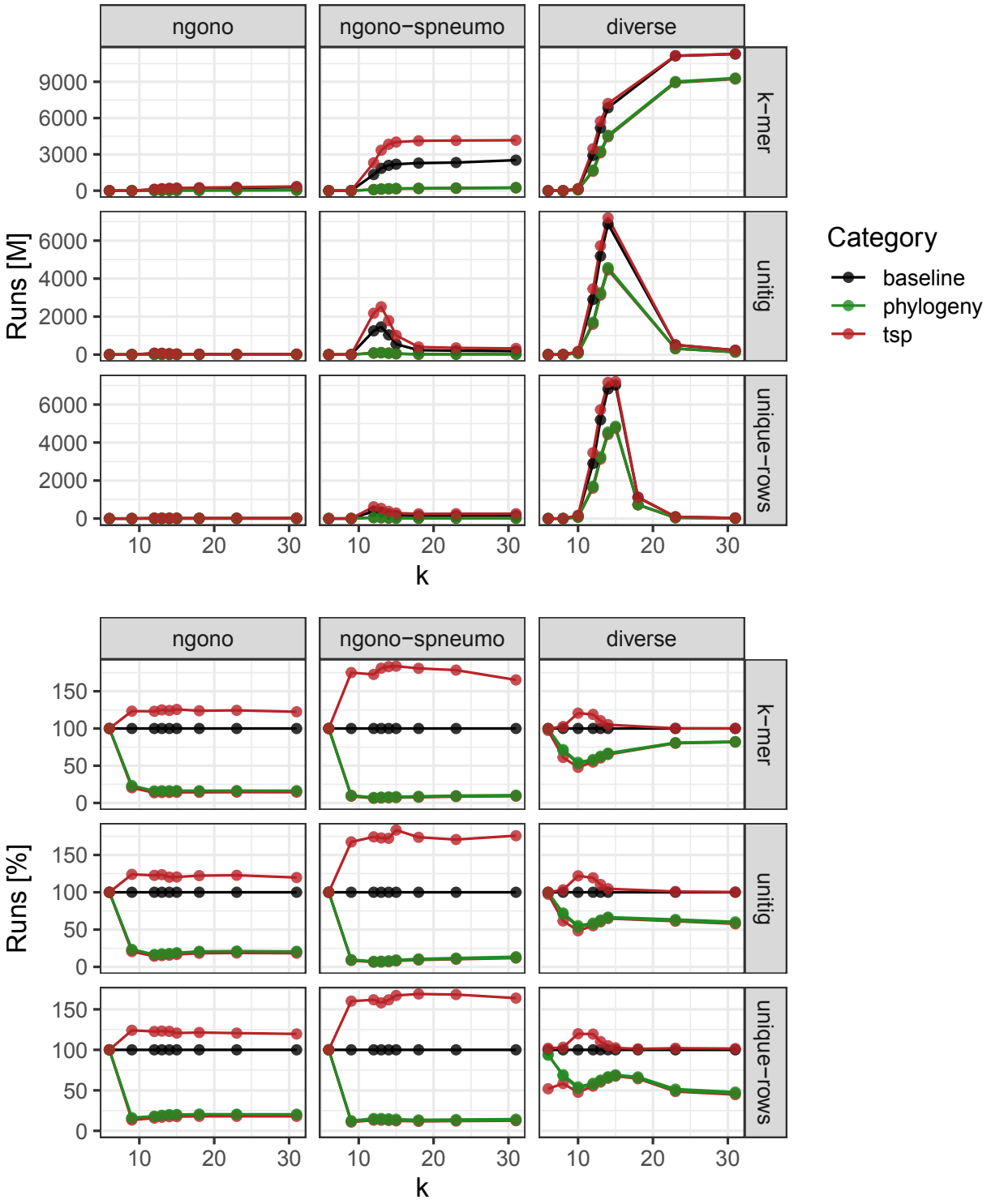


Figure 5.4: Impact of k-mer size on the compression performance. We evaluated how varying the k-mer length affects the total number of runs in RLE-compressed binary matrices across three datasets of size $N = 1000$ and three matrix types. For given value of k, we computed the run count under baseline, phylogeny-guided (NJ), and worst-case and optimal column orderings.

We found that the choice of k has a substantial impact on the size of the different matrix types and consequently their compressibility. We evaluated several values of k between 6 and 31; however, for the `diverse` dataset, the k-mer and unitig matrices became too computationally demanding for $k \in \{15, 18\}$. In these cases, the pipeline did not finish due to the time required to compute Hamming distances (Section 4.2.4). To compensate, we computed additional points around $k = 14$. Based on trends observed in the less diverse datasets, we expect the missing k-mer matrix values to lie between the results for $k = 14$ and $k = 23$. For the unitig matrix, we anticipate a peak within this range, similar to that seen in the unique-row matrix but with larger absolute values.

The results further confirm the robustness of phylogeny-guided ordering improvements. Across all datasets, matrix types, and k values, both NJ and UPGMA orderings remain extremely close to the optimal TSP solution. Even in cases where the matrices are dense or significantly violate ISM assumptions, they consistently approach the optimum within a few percentage points.

The three matrix types exhibit distinct behaviors as k increases. The k-mer matrices begin in a highly dense regime for small k, since short k-mers appear in almost all genomes, and their height grows rapidly until stabilizing around $k = 18$. The unitig matrices show an even sharper initial increase, where for small k, the de Bruijn graph is highly fragmented, producing many short unitigs, which inflate the matrix height. As k increases, these fragments merge into longer unitigs, and the total number of unitigs decreases again around $k = 18$. The unique-row matrices follow a similar pattern but with substantially smaller absolute sizes, peaking around $k = 15$.

Despite these differences in absolute matrix size, the relative compression performance measured against the randomized baseline shows much more stable behavior. After the dense, low-specificity regime at small k, the relative run counts settle quickly and remain nearly constant across the remaining k values, with only minor fluctuations in the `diverse` dataset. This suggests that once k is large enough for the matrices to reflect meaningful evolutionary structure, the relative advantage of different orderings becomes largely independent of the exact choice of k.

5.5 Pipeline performance evaluation

All experiments were executed on a dedicated compute node equipped with two 16-core AMD EPYC 7281 processors running at 2.1 GHz (SMT disabled), 128 GB of RAM, and 360 GB of local SSD storage. The node ran CentOS 7.8 and was managed by the PBS Pro 18.1 job scheduler. Jobs were submitted with a wall-time limit of 72 hours and exclusive access to 32 CPU cores and 120 GB of memory on a single node.

The runtimes of individual pipeline components varied substantially across datasets and matrix types, with the `diverse` dataset and k-mer matrix construction typically taking the longest times. Fulgor index construction typically required on the order of several tens of minutes to a few hours. Dumping the unitig and color-set files from the index took only a few minutes. Computing Hamming distances was one of the most time-consuming steps, ranging from several hours to multiple days depending on dataset size and matrix type. Unique-row matrices were generally faster to process, whereas unitig and k-mer matrices required streaming and checking memberships in the full `unitigs.fa` file. Phylogenetic tree inference with Attotree completed within a few minutes. Concorde exhibited highly variable runtimes: most TSP instances were solved within minutes to hours, but the largest instances occasionally required several days. The final RLE evaluation was computationally negligible, completing within seconds.

Disk usage was dominated by the Fulgor indexes, which reached 10–15 GB for the largest datasets. Compressed distance matrices occupied only tens of megabytes, and all other intermediate files were comparatively small. Peak memory usage during computation was not measured explicitly, but allocating 120 GB per job was always sufficient.

Two components of the pipeline represented bottlenecks for instances of 1 000 genomes. First, the distance computation step was a significant bottleneck, sometimes exceeding three days before parallelization was introduced. Although the task is conceptually simple, it is inherently time-consuming for datasets and k values producing many unitigs. Second, Concorde's exact branch-and-cut algorithm can require substantial time not only to find an optimal tour but also to verify its optimality. This verification phase often dominates the total runtime, especially for large instances with large distances. As a result, scaling the TSP-based ordering approach beyond a few thousand genomes may be challenging. In contrast, the remaining components of the pipeline scale more favorably, provided that Fulgor index construction remains tractable for larger datasets.

Chapter 6

Discussion and conclusion

In this thesis, we aimed to explore the theoretical foundations and practical behavior of phylogenetic compression, a technique that exploits evolutionary structure to improve the compressibility of large genomic collections. Our work combined mathematical modeling, algorithmic formulation, and empirical evaluation on real bacterial datasets, providing a unified view of when and why phylogeny-guided ordering yields near-optimal compression.

On the theoretical side, we introduced and formally characterized a framework for modeling phylogenetic compression. Within this framework, we proved that the simplified phylogenetic protocol (constructing a tree from pairwise distances and reordering data in left-to-right leaf order) achieves the optimal RLE compression. We further showed that the compression problem itself can be expressed as a discrete optimization task equivalent to solving the Traveling Salesperson Problem (TSP) under Hamming distances, showing that the general problem is NP-hard. Moreover, we defined the class of ISM-compliant matrices, a generalization of the binary matrices built on data following the Infinite Sites Model (ISM) that captures the essential structural properties required for phylogeny-based ordering to be optimal, and showed that several frequently used types of bioinformatics matrices belong to this class.

To assess how this theoretical model translates to real microbial data, we implemented an experimental evaluation pipeline using Snakemake. The workflow constructs k-mer, unitig, and unique-row matrices from genome assemblies, computes pairwise column distances, generates optimal and worst-case TSP orderings, infers phylogenetic trees using two distance-based methods, and evaluates the resulting RLE-compressed sizes. This pipeline enabled a systematic comparison across three bacterial datasets of varying diversity, and across multiple k-mer sizes, with experiments performed on subsets of up to 1000 genomes.

The empirical results demonstrate that many of the theoretical insights derived under ISM-like assumptions remain remarkably robust in practice. Even though real bacterial genomes violate the idealized model through homoplasy, recombination, and structural variation, their binary matrices still exhibit strong tree-like structure. As a consequence, phylogeny-based orderings consistently achieve compression close to the optimal TSP solution. In particular, the Neighbor-Joining algorithm (NJ) performs near-optimally not only for single-species datasets but also for more diverse collections, and basically across all our experiments. These findings suggest that bacterial genome space retains enough additive signal for simple distance-based methods to approximate the optimal ordering extremely well.

Interestingly, our experiments also show that biologically more accurate trees do not always yield better compression. For example, in the *N. gonorrhoeae* dataset, the RAxML-inferred tree performs worse than NJ. This highlights a key distinction between phylogenetic accuracy and compressive utility: the latter depends primarily on capturing local geometric relationships between genomes, a task that distance-based methods and even TSP heuristics can accomplish efficiently.

Our work was limited by several practical constraints. First, our protocol of phylogenetic compression is highly simplified and does not reflect the full complexity of methods used in practice. Another limitation is that our experimental evaluation focused on datasets of 1 000 genomes, which is negligible compared to large-scale genomic collections such as AllTheBacteria [4], where individual species may contain hundreds of thousands of genomes and the full collection spans thousands of species. It remains unclear how our findings would scale to such magnitudes, both in terms of computational tractability and the stability of compression gains. Furthermore, our theoretical framework relies on several simplifying assumptions about the data that do not fully reflect real-world genomic variation. Moreover, the use of RLE as our low-level compression model is

itself a strong simplification: RLE is rarely employed in practice in this form, but we adopted it for its analytical tractability and ease of implementation. A more realistic model could incorporate more complex compressors such as Lempel–Ziv.

Our results also point to several promising directions for further optimization. Because real data are not perfectly tree-additive, different leaf orderings of the same inferred tree can yield different compression outcomes, suggesting the potential for second-order improvements through branch rotations or local search. Moreover, our simplified RLE-based model captures only one aspect of practical compression tools. Extending the framework to incorporate row merging, alternative matrix layouts, or Lempel–Ziv–based compressors could bring the model closer to real systems such as Metagraph. Finally, integrating clustering steps or exploring hybrid phylogeny-TSP approaches may further enhance performance on highly diverse datasets.

Another interesting direction of future research is incorporating clustering into our model of phylogenetic compression. Given a collection of genomes, the first step would be to divide it into batches of comparable size, within which phylogeny inference and compression are performed independently. Even though we explored this direction, this proved more challenging than initially expected. Preliminary experiments indicated that meaningful clustering would require handling matrices of varying sizes, complicating both the modeling and the evaluation. In principle, clustering could be integrated into our pipeline by introducing additional dummy cities into the TSP formulation, allowing the solver to partition genomes into optimally sized clusters for a given number of clusters. However, implementing similar clustering for phylogenetic orderings would have required substantial additional effort, with uncertain computational costs. For these reasons, we ultimately chose not to implement clustering within the scope of this simplified model, although the idea remains conceptually feasible and may offer a promising direction for future work.

Overall, this work provides both a theoretical foundation and an empirical validation for phylogenetic compression. By showing that simple, scalable methods can achieve near-optimal results even under substantial deviations from ideal evolutionary assumptions, it strengthens the case for using evolutionary structure as a guiding principle in the design of compressed genomic data structures. Together, these findings offer the first mathematical explanation for the effectiveness of phylogeny-guided compression and indexing. They reveal how the evolutionary structure inherent in bacterial genomes can be leveraged to overcome computational hardness barriers and achieve near-optimal compression in practice. In the long term, these insights contribute to the broader goal of developing entropy-scaling algorithms capable of sublinear search and analysis on massive genomic collections, and they provide conceptual foundations that may guide the design of future data structures and algorithms for large-scale genomic analysis.

Bibliography

[1] Pratibha Mambatta Sankaranarayanan et al. “A review of next-generation sequencing technologies and their impact on clinical research: Assessing clinical efficacy and cost-effectiveness”. In: *Trends Curr Biol* 2.3 (July 2024).

[2] Gaye Lightbody et al. “Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application”. en. In: *Brief. Bioinform.* 20.5 (Sept. 2019), pp. 1795–1811.

[3] Paul A Kitts et al. “Assembly: a resource for assembled genomes at NCBI”. en. In: *Nucleic Acids Res.* 44.D1 (Jan. 2016), pp. D73–80.

[4] M Hunt et al. “AllTheBacteria - all bacterial genomes assembled, available and searchable”. In: *bioRxiv* (Mar. 2024).

[5] Zachary D Stephens et al. “Big Data: Astronomical or genomical?” en. In: *PLoS Biol.* 13.7 (July 2015), e1002195.

[6] Po-Ru Loh, Michael Baym, and Bonnie Berger. “Compressive genomics”. en. In: *Nat. Biotechnol.* 30.7 (July 2012), pp. 627–630.

[7] S McGinnis and T L Madden. “BLAST: at the core of a powerful and diverse set of sequence analysis tools”. In: *Nucleic Acids Res.* 32.Web Server issue (July 2004), W20–W25.

[8] K Břinda. *Growth of the NCBI Bacterial Assembly DB.* 2024.

[9] Karel Břinda et al. “Efficient and robust search of microbial genomes via phylogenetic compression”. en. In: *Nat. Methods* 22.4 (Apr. 2025), pp. 692–697.

[10] S Grabowski and T M Kowalski. “MBGC: Multiple Bacteria Genome Compressor”. In: *Gigascience* 11 (Jan. 2022), giab099.

[11] S Deorowicz, A Danek, and H Li. “AGC: compact representation of assembled genomes with fast queries and updates”. In: *Bioinformatics* 39.3 (Mar. 2023), btad097.

[12] Harun Mustafa et al. “MetaGraph-MLA: Label-guided alignment to variable-order De Bruijn graphs”. In: *bioRxiv* (Nov. 2022).

[13] J N Alanko et al. “Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes”. In: *Bioinformatics* 39 (June 2023), pp. i260–i269.

[14] J Fan, J Khan, and N P Singh. “Fulgor: a fast and compact k-mer index for large-scale matching and color queries”. In: *Algorithms Mol. Biol.* 19 (2024), p. 3.

[15] N Saitou and M Nei. “The neighbor-joining method: a new method for reconstructing phylogenetic trees”. In: *Mol. Biol. Evol.* 4.4 (1987), pp. 406–425.

[16] Kübra Eren, Nursema Taktakoğlu, and Ibrahim Pirim. “DNA sequencing methods: From past to present”. en. In: *Eurasian J. Med.* 54.Suppl1 (Dec. 2022), pp. 47–56.

[17] S Goodwin, J D McPherson, and W R McCombie. “Coming of age: ten years of next-generation sequencing technologies”. In: *Nat. Rev. Genet.* 17.6 (2016), pp. 333–351.

[18] Lin Liu et al. “Comparison of next-generation sequencing systems”. en. In: *J. Biomed. Biotechnol.* 2012 (July 2012), p. 251364.

[19] Shanika L Amarasinghe et al. “Opportunities and challenges in long-read sequencing data analysis”. en. In: *Genome Biol.* 21.1 (Feb. 2020), p. 30.

[20] Anthony Rhoads and Kin Fai Au. “PacBio Sequencing and its applications”. en. In: *Genomics Proteomics Bioinformatics* 13.5 (Oct. 2015), pp. 278–289.

[21] Phelim Bradley et al. “Ultrafast search of all deposited bacterial and viral genomic data”. en. In: *Nat. Biotechnol.* 37.2 (Feb. 2019), pp. 152–159.

[22] B D Ondov, T J Treangen, and P Melsted. “Mash: fast genome and metagenome distance estimation using MinHash”. In: *Genome Biology* 17 (2016), p. 132.

[23] R R Sokal and C D Michener. “A Statistical Method for Evaluating Systematic Relationships”. In: *University of Kansas Scientific Bulletin* 28 (1958), pp. 1409–1438.

[24] Alexandros Stamatakis. “RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies”. en. In: *Bioinformatics* 30.9 (May 2014), pp. 1312–1313.

[25] Nicholas J Croucher et al. “Rapid phylogenetic analysis of large samples of recombinant bacterial whole genome sequences using Gubbins”. en. In: *Nucleic Acids Res.* 43.3 (Feb. 2015), e15.

[26] L S Katz Katz et al. “Mashtree: a rapid comparison of whole genome sequence files”. In: *J. Open Source Softw.* 4.44 (2019), p. 1762.

[27] Larry L Peterson and Bruce S Davie. “End-to-end data”. In: *Computer Networks*. Elsevier, 2022, pp. 554–605.

[28] Christos H Papadimitriou. *Computational Complexity*. Upper Saddle River, NJ: Pearson, Nov. 1993.

[29] M Bläser. “Metric TSP”. In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. Boston, MA: Springer, 2008.

[30] N Christofides. “Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem”. In: *Operational Research Forum* 3 (2022), p. 20.

[31] Ch H Papadimitriou. “The Euclidean travelling salesman problem is NP-complete”. In: *Theoretical Computer Science* 4.3 (1977), pp. 237–244.

[32] J Ernvall, J Katajainen, and M Penttonen. “NP-completeness of the hamming salesman problem”. In: *BIT* 25 (1985), pp. 289–292.

[33] Dan Gusfield. “Efficient algorithms for inferring evolutionary trees”. In: *Networks* 21.1 (1991), pp. 19–28.

[34] Hans L Bodlaender et al. “The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs”. en. In: *Theor. Comput. Sci.* 244.1-2 (Aug. 2000), pp. 167–188.

[35] Itsik Pe’er et al. “Incomplete directed perfect phylogeny”. In: *SIAM J. Comput.* 33.3 (Jan. 2004), pp. 590–607.

[36] G F Estabrook, C S Johnson Jr, and F R McMorris. “A mathematical foundation for the analysis of cladistic character compatibility”. en. In: *Math. Biosci.* 29.1-2 (Jan. 1976), pp. 181–187.

[37] G F Estabrook, C S Johnson Jr, and F R Mc Morris. “An idealized concept of the true cladistic character”. en. In: *Math. Biosci.* 23.3-4 (Apr. 1975), pp. 263–272.

[38] Tamar Barzuza et al. “Computational problems in perfect phylogeny haplotyping: typing without calling the allele”. en. In: *IEEE/ACM Trans. Comput. Biol. Bioinform.* 5.1 (Jan. 2008), pp. 101–109.

[39] Christopher A Meacham. “Theoretical and computational considerations of the compatibility of qualitative taxonomic characters”. In: *Numerical Taxonomy*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 304–314.

[40] M Kimura. “The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations”. In: *Genetics* 61.4 (Apr. 1969), pp. 893–903.

[41] G A Watterson. “On the number of segregating sites in genetical models without recombination”. en. In: *Theor. Popul. Biol.* 7.2 (Apr. 1975), pp. 256–276.

[42] F Tajima. “Evolutionary relationship of DNA sequences in finite populations”. en. In: *Genetics* 105.2 (Oct. 1983), pp. 437–460.

[43] Laurent Excoffier et al. “Robust demographic inference from genomic and SNP data”. en. In: *PLoS Genet.* 9.10 (Oct. 2013), e1003905.

[44] Bo Peng and Marek Kimmel. “simuPOP: a forward-time population genetics simulation environment”. en. In: *Bioinformatics* 21.18 (Sept. 2005), pp. 3686–3687.

[45] Asger Hobolth, Marcy K Uyenoyama, and Carsten Wiuf. “Importance sampling for the infinite sites model”. en. In: *Stat. Appl. Genet. Mol. Biol.* 7.1 (Oct. 2008), Article32.

[46] P A P Moran. “Random processes in genetics”. en. In: *Math. Proc. Camb. Philos. Soc.* 54.1 (Jan. 1958), pp. 60–71.

[47] Franz Baumdicker et al. “Efficient ancestry and mutation simulation with msprime 1.0”. en. In: *Genetics* 220.3 (Mar. 2022).

[48] Richard R Hudson. “Generating samples under a Wright-Fisher neutral model of genetic variation”. en. In: *Bioinformatics* 18.2 (Feb. 2002), pp. 337–338.

[49] Kevin R Thornton. “A C++ template library for efficient forward-time population genetic simulation of large populations”. en. In: *Genetics* 198.1 (Sept. 2014), pp. 157–166.

[50] Laurent Excoffier et al. “Fastsimcoal2: Demographic inference under complex evolutionary scenarios”. en. In: *Bioinformatics* 37.24 (Dec. 2021), pp. 4882–4885.

[51] Benjamin C Haller and Philipp W Messer. “SLiM 3: Forward genetic simulations beyond the Wright-Fisher model”. en. In: *Mol. Biol. Evol.* 36.3 (Mar. 2019), pp. 632–637.

[52] Stefano G Giulieri et al. “A multi-hospital, clinician-initiated bacterial genomics programme to investigate treatment failure in severe Staphylococcus aureus infections”. en. In: *Nat. Commun.* 16.1 (May 2025), p. 4869.

[53] R R Hudson and N L Kaplan. “Statistical properties of the number of recombination events in the history of a sample of DNA sequences”. en. In: *Genetics* 111.1 (Sept. 1985), pp. 147–164.

[54] Dan Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge, England: Cambridge University Press, Mar. 2011.

[55] Peter Buneman. “The recovery of trees from measures of dissimilarity”. In: *Mathematics in the Archaeological and Historical Sciences*. Ed. by F R Hodson, D G Kendall, and P Tautu. Edinburgh: Edinburgh University Press, 1971, pp. 387–395.

[56] D Applegate et al. *Concorde TSP Solver*. Computer software available from University of Waterloo. 1997.

[57] O Gascuel and M Steel. “Neighbor-Joining Revealed”. In: *Mol. Biol. Evol.* 23.11 (2006), pp. 1997–2000.

[58] Johannes Köster and Sven Rahmann. “Snakemake-a scalable bioinformatics workflow engine”. en. In: *Bioinformatics* 34.20 (Oct. 2018), p. 3600.

[59] K Břinda. *attotree: Software for Efficient and Robust Microbial Genome Searches via Phylogenetic Compression*. `https://github.com/karel-brinda/attotree`. 2024.

[60] Howe K., Bateman A., and R Durbin. “QuickTree: building huge Neighbour-Joining trees of protein sequences”. In: *Bioinformatics* 18 (Nov. 2002), pp. 1546–1547.

[61] J A Studier and K J Keppler. “A note on the neighbor-joining algorithm of Saitou and Nei”. In: *Mol. Biol. Evol.* 5.6 (1988), pp. 729–731.

[62] R Durbin et al. *Biological sequence analysis*. Online edition. Cambridge, England: Cambridge University Press, Sept. 1998.

[63] M K Kuhner and J Felsenstein. “A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates”. In: *Mol. Biol. Evol.* 11.3 (1994), pp. 459–468.

[64] D Bryant. “On the Uniqueness of the Selection Criterion in Neighbor-Joining”. In: *J. Classif.* 22 (2005), pp. 3–15.

[65] Y Pauplin. “Direct Calculation of a Tree Length Using a Distance Matrix”. In: *J. Mol. Evol.* 51 (2000), pp. 41–47.

[66] R Desper and O Gascuel. “The Minimum Evolution Distance-Based Approach to Phylogenetic Inference”. In: *Mathematics of Evolution & Phylogeny*. Ed. by O Gascuel. Oxford, UK: Oxford University Press, 2005, pp. 1–32.

Appendices

A Neighbor-joining algorithm (NJ)

Here we describe the NJ algorithm in detail and summarize some results of its mathematical analysis that was done in the past.

Given a set of objects and their pairwise distances, the algorithm iteratively connects pairs of neighboring leaves, creating subtrees until an unrooted tree emerges. The core of the method is therefore the formula that determines the pair of neighboring leaves to be connected by a parental node. It is not sufficient to pick the leaves with the smallest distance, as it is demonstrated in Figure A.1. Instead, the neighboring leaves are determined as the pair with the smallest value of the criterion

$$\begin{aligned} D_{ij} &= d_{ij} - (r_i + r_j)\,, \\ r_i &= \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}\,, \end{aligned} \tag{1}$$

where L is the set of leaves and d_{ij} is the distance between the leaves i and j. This formulation ([61]) is not the one originally formulated by Saitou and Nei, however, they were proven equivalent ([57]). Studier and Keppler also provided the proof of the connection between the criterion D_{ij} and the leaves' neighborhood.

Theorem 8 ([15], [61])**.** If i and j are leaves chosen so that D_{ij} is minimal, then i and j are neighbors.

An iteration of the NJ algorithm runs as illustrated in Figure A.2. Once the algorithm identifies a neighboring pair of leaves i and j, it defines a parental node k with the corresponding distances

$$d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j), d_{jk} = d_{ij} - d_{ik}\,, \tag{2}$$

and sets the new node's distances to other leaves to

$$d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})\,. \tag{3}$$

The leaves i and j are then removed from the list of leaves L and k is added, representing a new leaf for the next iteration. The whole process is described as Algorithm 1.

The algorithm is performing well with both simulated and real data ([15], [63]), and the selection criterion D_{ij} was proven to be unique and consistent among others that are linear, permutation equivariant, statistically

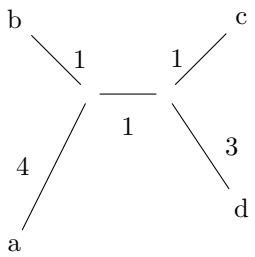


Figure A.1: Example of an unrooted tree where the pair of leaves connected by the shortest path (b and c with $d_{bc} = 3$) is not a neighboring one.

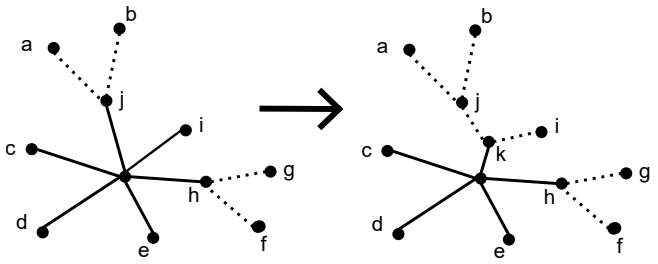


Figure A.2: One iteration of the NJ algorithm. Nodes j and i are identified as neighbors, and a parental node k is created. Resolved nodes i and j are then removed from the set of current leaves and k is added for the next iteration.

Algorithm 1: Neighbor-joining (adopted from [62])

Data: set of leaves and their pairwise distances d_{ij}
Result: set of nodes and branch lengths of the corresponding unrooted tree
Initialization:
Define a set of leaves L containing provided data;
Define a set of all nodes T, put $T = L$;
while *there are more than two leaves in* L **do**
Determine the leaves i, j for which D_{ij} is minimal;
Define a new node k and set $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in L$;
Add k to T as parent of i, j with edge lengths $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$, $d_{jk} = d_{ij} - d_{ik}$;
Add k to the set of leaves and remove i and j;
end
Termination:
Connect the two remaining leaves m and n with an edge of length d_{mn};

consistent and based solely on distance ([64]). However, it is not immediately clear from Equation (1) what is the property that is being minimized by the NJ algorithm. It took more than ten years from the NJ introduction before the question was rigorously answered. Gascuel and Steel provide an insightful review of the subject ([57]).

The core of the answer lies in so called generalized Pauplin formula, which uses a weighted sum to obtain the total branch length in a tree.

Definition 14 ([57], [65])**.** Let i, j be leaves in a weighted tree. Consider the path from i to j in the tree with n interior nodes. Let o_k denote the number of branches associated with the interior node k. The weight w_{ij} is defined as

$$w_{ij} = \frac{1}{\prod_{k=1}^{n}(o_k - 1)}\,.$$

Theorem 9 ([57], [65])**.** Let l be the total sum of all branch lengths of a weighted tree. Then $l = \sum_{i,j} w_{ij} d_{ij}$.

Theorem 10 ([57], [66])**.** The NJ method, as defined by equations 1, 2, and 3 selects at each step as neighbors that pair of current leaves, which most decreases the whole tree length, as computed using the generalized Pauplin formula from Theorem 9.

B Table of datasets

Name	#genomes	#species	#distinct 31-mers	Description
ngono	1000	1	4,117,063	High-quality genomes of *Neisseria gonorrhoeae*. `https://zenodo.org/records/15367750/files/part_54.tar`
spneumo	1000	1	16,268,711	High-quality genomes of *Streptococcus pneumoniae*. `https://zenodo.org/records/15367750/files/part_85.tar`
ngono-spneumo	1000	2	16,900,806	A random subset of 500 *ngono* and 500 *spneumo* genomes.
diverse	1000	539	3,009,067,743	A random subset of 1000 genomes from all dustbins of the phylogenetically compressed 661k collection. `https://zenodo.org/records/15367750/`
ngono-rase	1102	1	4,179,242	Draft genome assemblies from Illumina HiSeq reads from the RASE database. `https://github.com/karel-brinda/rase-db-ngonorrhoeae-gisp`

Table 1: Genome collections used in our experiments.

C Comparison of alternative RLE compression strategies

To evaluate whether our choice of RLE compression strategy influences the experimental results, we compared the row-wise approach used in our theoretical model (where each row begins a new run) with an alternative strategy that linearizes the entire matrix and applies RLE globally (Figures C.3 to C.5). Across all three main experiments, the relative differences between ordering methods remain highly consistent under both compression styles.

The only noticeable deviation appears in the `diverse` dataset. In the experiment varying dataset size, sparse rows introduce an additive constant that uniformly increases the total number of runs under the row-wise RLE strategy. In the experiment varying k-mer size, the difference between the two run types in the k-mer matrix grows with increasing k, reflecting the fact that the matrices become progressively sparser. This is the only setting in which the two strategies do not differ by a simple uniform shift. Nevertheless, the overall shape and behavior of the curves remain the same, and the gap would eventually converge to the height of the matrix once most k-mers become unique. For the remaining datasets, the differences between the two RLE strategies are negligible.

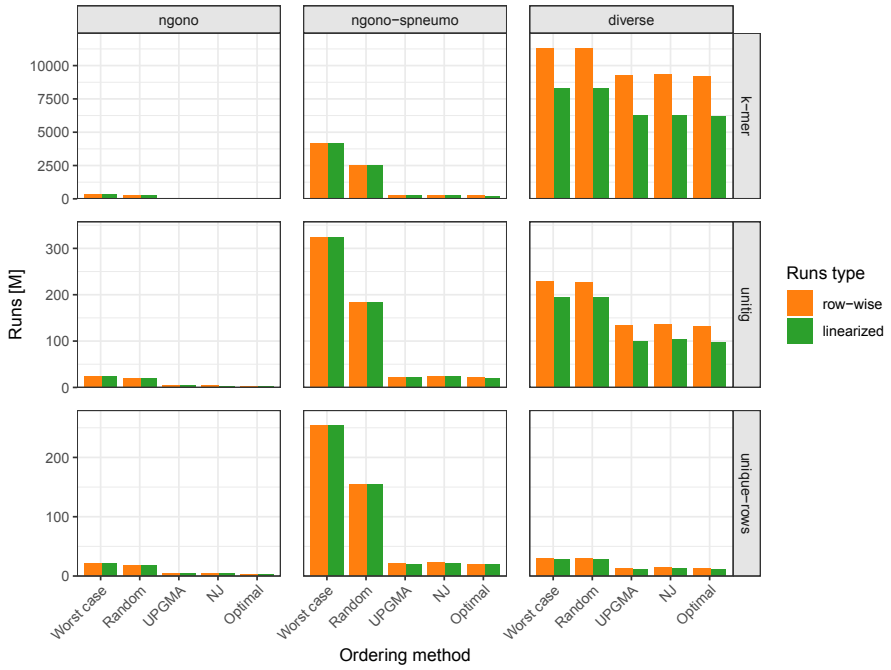


Figure C.3: Comparison of RLE compression strategies across orderings and datasets. Comparison of row-wise and linearized global RLE in the ordering experiment shows that relative differences between optimal, random, and phylogeny-guided orderings remain stable across both compression strategies.

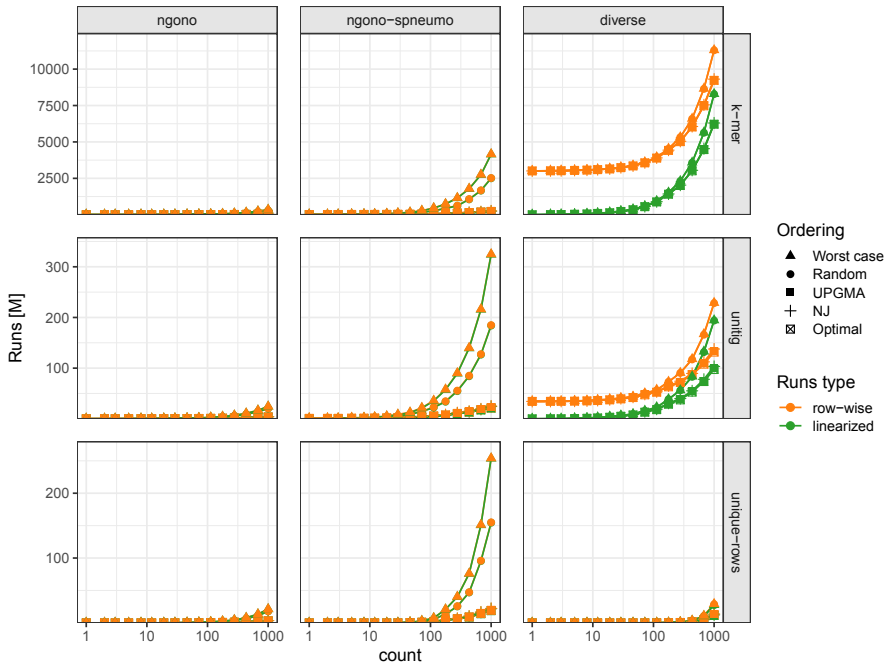


Figure C.4: Comparison of RLE compression strategies across dataset sizes. For increasing numbers of genomes, both RLE strategies yield nearly identical trends, with only a uniform shift in the `diverse` dataset due to sparse rows.

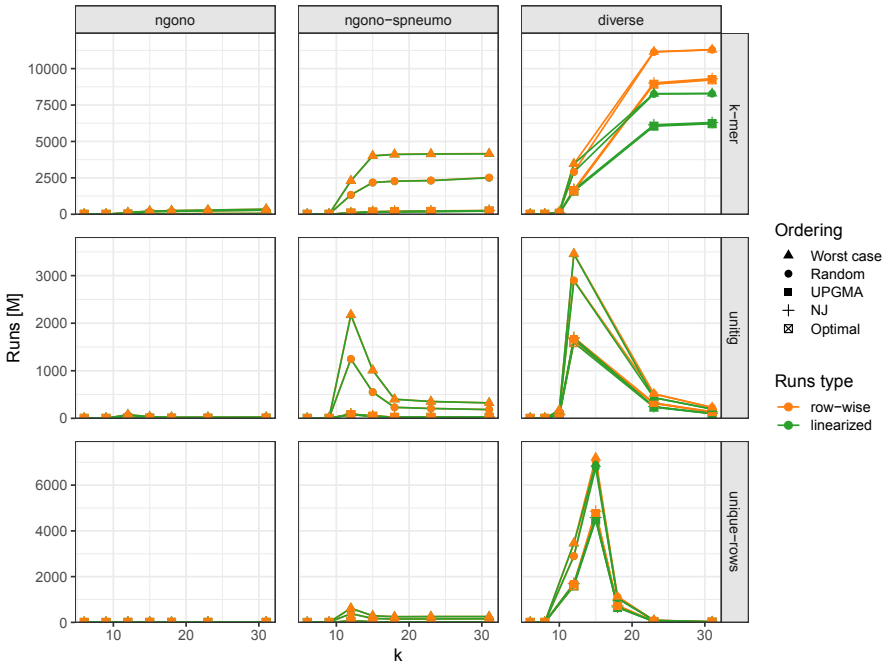


Figure C.5: Comparison of RLE compression strategies across k-mer sizes. Across all tested values of k, the two RLE approaches produce consistent relative ordering performance. The differences in the `diverse` dataset are caused by sparse rows, since many k-mers are unique.